

ALIGN.CPP

11/05/98
09/186962
JCS03 U.S. PTO

```

scale_increment=pow(1.0/(double)START_RADIUS, 1.0/(double)lp_sampling);
for(i=0;i<lp_sampling;i++){
    radius[i] = (START_RADIUS*(double)dim2) * pow(scale_increment, (double)i);
}

```

```

pout = out;
for(theta=0.0,j=0;j<lp_sampling;j++,theta += (PI/lp_sampling)){
    dx = cos(theta);
    dy = sin(theta);
    radius = radius;
    pout = tout[j];
    for(i=0;i<lp_sampling;i++){
        x = (double)dim2 * radius * dx;
        y = (int)x;
        xx = (int)x;
        yy = (int)y;
        frax = x - (double)xx;
        fracy = y - (double)yy;
        pin = sin(yy*dim + xx);
        *pout += (float) ( (1.0-frax)*(1.0-fracy)* (double)*(pin++) );
        pin += (float) ( frax*(1.0-fracy)* (double)*pin );
        pin += (dim-1);
        *pout += (float) ( (1.0-frax)*fracy* (double)*(pin++) );
        *pout += (float) ( frax*fracy * (double)*pin );
        pout += lp_sampling;
    }
}

```

```

/* now filter it along the scale axis */
/* this generally increases the peak to noise ratio in finding the proper scale rotation */
for(i=0;i<lp_sampling;i++){
    pout = ftemp;
    for(j=0;j<lp_sampling;j++){
        *pout = (float)0.0;
        for(k=-((LOG_MOV_AVG/2),k<=((LOG_MOV_AVG/2),k++){
            jj=j+k;
            if(jj<0)jj=0;
            else if(jj>=lp_sampling)jj=lp_sampling-1;
            *pout += out[i+j]*lp_sampling;
        }
        *pout += (float)LOG_MOV_AVG;
    }
    pin = ftemp;
    pout = tout[i];
    for(j=0;j<lp_sampling;j++){
        *pout = (float)0.0;
        for(k=-((LOG_SMOOTH/2),k<=((LOG_SMOOTH/2),k++){
            jj=j+k;
            if(jj<0)jj=0;
            else if(jj>=lp_sampling)jj=lp_sampling-1;
            *pout += out[i+j]*lp_sampling;
        }
        *pout += (float)LOG_SMOOTH;
    }
    memcpy(tout[i],ftemp,lp_sampling*sizeof(float) );
}
return(1);
}

```

```

float get_median(float(float *array,int xdim,int ydim,int high_x,int high_y,
float *x_offset,float *y_offset){
    float *median[3],*py,*px,ratio,value;
    int j,ftemp,k,ktemp;
    ymedian[0]=ymedian[1]=ymedian[2]=(float)0.0;
    xmedian[0]=xmedian[1]=xmedian[2]=(float)0.0;
    py = ymedian;
    for(j=1;j<2;j++){
        jtemp = high_y+j;
        if(jtemp < 0)jtemp=ydim-1;
        else if(jtemp==ydim)jtemp=0;
        px = xmedian;
        for(k=1;k<2;k++){
            kttemp = high_x+k;

```

```

float get_2d_median(float *array,int xdim,int ydim,int high_x,int high_y,
float *x_offset,float *y_offset){
    float *median[3],*py,*px,ratio,value;
    int j,ftemp,k,ktemp;
    ymedian[0]=ymedian[1]=ymedian[2]=(float)0.0;
    xmedian[0]=xmedian[1]=xmedian[2]=(float)0.0;
    py = ymedian;
    for(j=1;j<2;j++){
        jtemp = high_y+j;
        if(jtemp < 0)jtemp=ydim-1;
        else if(jtemp==ydim)jtemp=0;
        px = xmedian;
        for(k=1;k<2;k++){
            kttemp = high_x+k;

```

```

if(ktemp < 0)ktemp=xdim-1;
else if(ktemp==xdim)ktemp=0;
*py += array[jtemp*xdim+ktemp];
*(px++) += array[jtemp*xdim+ktemp];
}
py++;
}
/* now find median values */
ratio = get_median(float(ymedian);
*y_offset = (float)high_y + ratio;
ratio = get_median(float(xmedian);
*x_offset = (float)high_x + ratio;
value = (xmedian[0]+xmedian[1]+xmedian[2])/(float)3.0;
return(value);
}

```

```

/* this is the fft window profile for mitigating edge effects; change to other windows if
their better */
/* or...., maybe certain windows are better for certain tasks, e.g., log polar vs. straight
correlation */
int load_windowing_function(int dim,float *window){
    int i;
    double step,x,y;
    step = 2.0*PI / (double)(dim+1);
    for(i=0,x=step;i<dim;i++,x+=step){
        y = (1.0 - cos(x))/2.0;
        window[i] = (float)sqrt(y);
    }
    return(1);
}

```

```

int window_id_vector(
float *array,
int data_length,
int full_length
){
    int i;
    float *parray,*pwindow;
    float *window_function = new float(data_length);
    load_windowing_function(data_length,window_function);
    parray = array;
    pwindow = window_function;
    for(i=0;i<data_length;i++){parray++} = *pwindow++;
    if(full_length != data_length){
        for(i=0;i<(full_length - data_length),i++){parray++} = (float)0.0;
    }
    delete [] window_function;
    return(1);
}

```

```

/* this module specifically designed for the rough thumbnail registration
in an earlier version of this routine, I performed bi-linear interpolation
on the pixels, but now think this is overkill because of the later refinement
anyway, who knows */
int rotate_scale_translate_image(
float *out,
int outdim,
float *in,
int indim,
int orig_xdim,
int orig_ydim,
int downsample,
float rotation,
float scale
)
{
    int i,j,xx,yy;
    float a_const,b_const,x,y,dx,dy,*pout;
    float middle_in_x, middle_in_y,middle_out;
    /* make sure to place the center of the original array at the center of
the output array; this helps later translation bookkeeping */
    middle_in_x = (float)(orig_xdim - downsample)/(float)downsample/(float)2.0;
    middle_in_y = (float)(orig_ydim - downsample)/(float)downsample/(float)2.0;
    middle_out = (float)(outdim-1)/(float)2.0;
    rotation = -rotation; // who can keep track of CW and CCW anyway???
    a_const = (float)cos((double)rotation*PI/180.0)*scale;
    b_const = (float)sin((double)rotation*PI/180.0)*scale;
    dx = a_const;
    dy = b_const;
    pout = out;
    for(i=0;i<outdim;i++){
        x = middle_in_x - a_const*middle_out + b_const*(middle_out-(float)i) + (float)0.5;
        y = middle_in_y - b_const*middle_out - a_const*(middle_out-(float)i) + (float)0.5;

```

```

highest = *preall;
x_off[i] = k;
y_off[i] = j;
}
preall++;
}
}

/* step through the found candidates, finding inter-sample values for the peak location */
for(i=0; i<number_candidates; i++){
    ymedian[0]=ymedian[1]=ymedian[2]=((float)0.0;
    xmedian[0]=xmedian[1]=xmedian[2]=((float)0.0;
    py = ymedian;
    for(j=-1; j<2; j++){
        jtemp = y_off[i]+j;
        if(jtemp < 0) jtemp=dim-1;
        else if(jtemp==dim) jtemp=0;
        px = xmedian;
        for(k=-1; k<2; k++){
            ktemp = x_off[i]+k;
            if(ktemp < 0) ktemp=dim-1;
            else if(ktemp==dim) ktemp=0;
            *py += reall[jtemp*dim+ktemp];
            *px += reall[jtemp*dim+ktemp];
        }
        py++;
    }
    /* now find median values */
    ratio = get_median_float(ymedian);
    y_offset[i] = (float)dim2 - ((float)y_off[i] + ratio);
    ratio = get_median_float(xmedian);
    x_offset[i] = (float)dim2 - ((float)x_off[i] + ratio);
    value[i] = reall[x_off[i] + dim*y_off[i]];
}
return(i);

/* simple sub-routine for direct_registration
int get_working_dimension(
int alignment_mode,
int xdim1,
int ydim1,
int xdim2,
int ydim2,
int *downsample
){
    int highest=xdim1, go=1, fftdim;
    if(ydim1>highest) highest = ydim1;
    if(xdim2>highest) highest = xdim2;
    if(ydim2>highest) highest = ydim2;
    switch(alignment_mode){
        case 0: // no downsampling
            *downsample = 1;
            fftdim = 1;
            while( go ){
                if( highest > fftdim ){
                    fftdim*=2;
                } else go = 0;
            }
            break;
        // nominal downsampling
        case 1:
            *downsample = ((highest-1)/NOMINAL_DOWNSAMPLE_DIM)+1;
            fftdim = NOMINAL_DOWNSAMPLE_DIM;
            break;
        // super downsampling
        case 2:
            *downsample = ((highest-1)/SUPER_DOWNSAMPLE_DIM)+1;
            fftdim = SUPER_DOWNSAMPLE_DIM;
            break;
    }
    return(fftdim);
}

/* another sub-routine for direct_registration
int copy_downsample_window(
unsigned char *in,
int xdim,
int ydim,
float *out,
int outdim,
int outdim,
int *downsample
){
    unsigned char *pin;
    int i,j;
}

```

```

out *pout; *pwindow, normalize;

```

```

pin = in;
memset(out, 0, outdim*outdim*sizeof(float));
for(i=0; i<ydim; i++)
    pout = out + (i*ydim) * outdim;
for(j=0; j<xdim; j++)
    pout[j*downsample] += (float)*(pin++);
}

// normalize it for downsampling
if(downsample > 1){
    xdim = 1 + (xdim-1)/downsample;
    ydim = 1 + (ydim-1)/downsample;
    normalize = (float)downsample * (float)downsample;
    for(i=0; i<ydim; i++){
        pout = out + i * outdim;
        for(j=0; j<xdim; j++){
            *(pout++) /= normalize;
        }
    }
}

if(WINDOW_ORIGINALS){
    float *window_function = new float[outdim];
    load_window_function(xdim, window_function);
    pout = out;
    for(i=0; i<ydim; i++){
        pwindow = window_function;
        for(j=0; j<xdim; j++){
            *(pout++) *= *pwindow++;
        }
        pout += (outdim-xdim);
    }
    load_window_function(ydim, window_function);
    pout = out;
    for(i=0; i<ydim; i++){
        pwindow = window_function;
        for(j=0; j<xdim; j++){
            *(pout++) *= *pwindow;
        }
        pout += (outdim-xdim);
    }
    delete [] window_function;
}

return(1);
}

int fourier_mellin_transform(
    float *in,
    float *ftemp,
    int dim,
    float *out
){
    int i, j;
    float *pout, *pwindow;

    convert_to_magnitude(ftemp, in, dim);
    log_polar_remap(ftemp, out, dim);
    if(WINDOW_LOGPOLAR_LOG){
        float *window_function = new float(lp_sampling);
        load_window_function(lp_sampling, window_function);
        pout = out;
        for(i=0; i<lp_sampling; i++){
            pwindow = window_function;
            for(j=0; j<lp_sampling; j++){
                *(pout++) *= *pwindow;
            }
        }
        delete [] window_function;
    }
    return(1);
}

int get_best_candidate(
    int number_candidates,
    float *ftemp,
    int dim,
    float *in,
    float *out,
    int xdim,
    int ydim,
    int xdim_orig,
    int ydim_orig,
    int downsample,

```

```

float *rotation,
float *scale,
float *x_trans,
float *y_trans,
float *template_real
){
    int i, highest_i, j;
    float highest = -(float)1e20, xtrans, ytrans, value;

    for(i=0; i<number_candidates; i++){
        for(j=0; j<2; j++){
            // rotate and scale suspect real image into ftemp *
            rotate_scale_translate_image(ftemp, dim, in, xdim, ydim, xdim_orig, ydim_orig,
            downsample, rotation[i]+(float)j*(float)180.0, scale[i]);
            realfft2d_in_place(ftemp, bits, 0, wr, wl);
            gmf(template_real, ftemp, dim, bits, i, xtrans, ytrans, &value, 1);
            if(value > highest){
                highest = value;
                highest_i = i;
                if(j==1) rotation[i] += (float)180.0;
                xtrans[i] = xtrans;
                ytrans[i] = ytrans;
            }
        }
        rotation[0] = rotation[highest_i];
        scale[0] = scale[highest_i];
        x_trans[0] = x_trans[highest_i];
        y_trans[0] = y_trans[highest_i];
    }
    return(1);
}

double log_id_remap(
    float *in,
    float *out,
    int dim
){
    int i, dim2 = dim/2, xx;
    float *pin, *pout;
    double radius, fracc;
    double scale_increment_id;

    scale_increment_id = pow( 1.0/(double)START_RADIUS_ID, 1.0/(double)dim);
    pout = out;
    for(i=0; i<dim; i++){
        radius = (START_RADIUS_ID*(double)dim2) * pow(scale_increment_id, (double)i);
        xx = (int)radius;
        fracc = radius - (double)xx;
        pin = &in[xx];
        *pout = (float) ( (1.0-fracc) * (double)*(pin++) );
        *(pout++) += (float) ( fracc * (double)*pin );
    }
    return(scale_increment_id);
}

int gmf_id(
    float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset
){
    int i, highest_i;
    float *preal1, *preal2, *pimaginary1, *pimaginary2;
    float mag1, mag2, dot, dot_cross, median[3], highest_ratio, ftmp;

    // calculate phase differences and reload them into real and imaginary1 */
    // keep phase differences to PI to -PI */
    preal1 = real1, pimaginary1 = imaginary1;
    preal2 = real2, pimaginary2 = imaginary2;
    for(i=0; i<dim; i++){
        mag1 = (float)sqrt( (double) (*preal1 * *preal1 + *pimaginary1 * *pimaginary1) );
        mag2 = (float)sqrt( (double) (*preal2 * *preal2 + *pimaginary2 * *pimaginary2) );
        if(mag1 == (float)0.0) mag1 = (float)SMALL;
        if(mag2 == (float)0.0) mag2 = (float)SMALL;
        dot = (*preal1 * *preal2 + *pimaginary1 * *pimaginary2) / (mag1/mag2);
        dot_cross = (float)1.0 - dot;
        if(dot < (float)0.0) dot = (float)0.0;
        dot_cross = (float)sqrt( (double)dot );
        dot = *preal1 * *pimaginary2 - *pimaginary1 * *preal2;
        if(dot_cross < (float)0.0) dot_cross = -(float)1.0;
        else dot_cross = (float)1.0;
        ftmp = mag2;
        dot_cross = dot_cross * ftmp;
        *(preal1++) = dot;
    }
}

```

```

    * (pimaginary1++) = cross*dott;
}

fft(real1,imaginary1,bits,1,wr,wi,1);

/* search for highest value, then median find the center */
highest = -(float)1e20;
preall = real1;
for(i=0;i<dim;i++){
    if(*preall > highest){
        highest = *preall;
        highest_i = i;
    }
    preall++;
}

if(highest_i == 0){
    median[0] = real1[dim-1];
    median[1] = real1[0];
    median[2] = real1[1];
}
else if(highest_i == (dim-1)){
    median[0] = real1[dim-2];
    median[1] = real1[dim-1];
    median[2] = real1[0];
}
else {
    median[0] = real1[highest_i-1];
    median[1] = real1[highest_i];
    median[2] = real1[highest_i+1];
}

ratio = get_median_float(median);
*offset = (float)highest_i * ratio;
if (*offset > (float)dim/2.0) *offset -= (float)dim;
return(1);
}

int refine_axis(
    unsigned char *template,
    int template_xdim,
    int template_ydim,
    unsigned char *suspect,
    int suspect_xdim,
    int suspect_ydim,
    float *x,
    float *y,
    int which
){
    unsigned char *pauspect;
    int i,j,highest,fftdim,bits,xx,yy,xdim,ydim;
    float x0,x1,x2,y0,y1,y2,*pauspect_integral,*template_integral;
    float scan_x,scan_y,jump_x,jump_y,current_x,current_y;
    float scale,translation_xdistance,ydistance,suspect_dc,template_dc,frac;
    double scale_increment_id;

    /* first convert the y axis version to the x axis version */
    x0 = x[0]; y0 = y[0];
    if(which){
        x1 = x[2]; y1 = y[2];
        x2 = x[1]; y2 = y[1];
        xdim = suspect_ydim;
        ydim = suspect_xdim;
    }
    else {
        x1 = x[1]; y1 = y[1];
        x2 = x[2]; y2 = y[2];
        xdim = suspect_xdim;
        ydim = suspect_ydim;
    }

    /* determine the next highest power of two above higher of the two suspect axes */
    if(suspect_xdim > suspect_ydim) highest = suspect_xdim;
    else highest = suspect_ydim;
    bits = 1 + (int)(log((double)highest - 0.5) / log(2.0));
    fftdim = (int)pow(2.0,(double)bits + 0.0000001);

    float *template_integral = new float[fftdim];
    float *suspect_integral = new float[fftdim];
    float *template_integral_imaginary = new float[fftdim];
    float *suspect_integral_imaginary = new float[fftdim];
    float *template_integral_copy = new float[fftdim];
    float *suspect_integral_copy = new float[fftdim];

    /* load suspect integral waveform */
    pauspect_integral = suspect_integral;
    for(j=0;j<fftdim;j++){ *pauspect_integral++ = (float)0.0;
        if(!which){
            pauspect = suspect;

```

```

convert to magnitude_id inplace(suspect_integral, suspect_integral_imaginary, fftdim);
convert to magnitude_id inplace(template_integral, template_integral_imaginary, fftdim);
// next routine places output inot_integral_imaginary, template_integral_imaginary, fftdim;
scale_increment_id = log10 remap(suspect_integral, suspect_integral_imaginary, fftdim);
scale_increment_id = log10 remap(template_integral, template_integral_imaginary, fftdim);
// copy output back into fundamental array and zero out imaginary
memcpy(suspect_integral, suspect_integral_imaginary, sizeof(float)*fttdim);
memcpy(template_integral, template_integral_imaginary, sizeof(float)*fttdim);
memset(suspect_integral_imaginary, 0, sizeof(float)*fttdim);
memset(template_integral_imaginary, 0, sizeof(float)*fttdim);
// now do the 1d fourier mullin trot
window_id vector(template_integral, fftdim, fftdim);
window_id vector(suspect_integral, fftdim, fftdim);
fft(suspect_integral, suspect_integral_imaginary, bits, 0, wr, wi, 1);
fft(template_integral, template_integral_imaginary, bits, 0, wr, wi, 1);
/* gmf id to find any small scaling difference between the two */
gmf_id(suspect_integral, suspect_integral_imaginary, template_integral,
template_integral_imaginary, fftdim, bits, fscale);
//scale = (float)0.6; // slight damping factor
scale = (float)pow(scale_increment_id, (double)scale);
// update the x's and y's
xdistance = (x1-x0);
ydistance = ((float)1.0 - scale);
ydistance = (y1-y0);
ydistance = (y1-y0);
x[3] += xdistance; y[3] += ydistance;
x[4] += xdistance/(float)2.0; y[4] += ydistance/(float)2.0;
if(which){
    x[2] += xdistance; y[2] += ydistance;
    x1 = x[2]; y1 = y[2];
}
else {
    x1 += xdistance; y[1] += ydistance;
    x1 = x[1]; y1 = y[1];
}
// now with the new scale information, perform a gmf on the original and its rescaled
counterpart */
template_integral = template_integral;
scale = (float)1.0 / scale;
float lllast;
for(i=0, current_x=(float)0.0; i<xdim; i++, current_x+=scale) {
    xx = (int)current_x;
    if(xx >= xdim-1){p(template_integral++) = lllast;
        else {
            frac = current_x - (float)xx;
            *template_integral = ((float)1.0-frac) * template_integral_copy(xx);
            *p(template_integral++) = frac * template_integral_copy(xx+1);
        }
        lllast = *p(template_integral-1);
    }
}
// window the new scaled array; other one should be copy of windowed original
memcpy(suspect_integral, suspect_integral_copy, sizeof(float)*fttdim);
window_id vector(template_integral, xdim, fftdim);
window_id vector(suspect_integral, xdim, fftdim);
memset(suspect_integral_imaginary, 0, sizeof(float)*fttdim);
memset(template_integral_imaginary, 0, sizeof(float)*fttdim);
fft(suspect_integral, suspect_integral_imaginary, bits, 0, wr, wi, 1);
fft(template_integral, template_integral_imaginary, bits, 0, wr, wi, 1);
// now find the translation
gmf_id(suspect_integral, suspect_integral_imaginary, template_integral,
template_integral_imaginary, fftdim, bits, ftranslation);
// adjust x and y accordingly
translation = (float)0.5; // I think this accounts for the fact that scaling has changed
origins????? very kludge
scan_x = translation;
scan_y = translation;
x[0] += scan_x; y[0] += scan_y;
x[1] += scan_x; y[1] += scan_y;
x[2] += scan_x; y[2] += scan_y;
x[3] += scan_x; y[3] += scan_y;
x[4] += scan_x; y[4] += scan_y;
delete [] template_integral;
delete [] suspect_integral;
delete [] template_integral_imaginary;
delete [] suspect_integral_imaginary;
delete [] template_integral_copy;
delete [] suspect_integral_copy;
return(0);
}
float refined_rotation(

```

```

float *x,
float *y,
unsigned char *suspect,
int suspect_xdim,
int suspect_ydim,
unsigned char *template,
int template_xdim,
int template_ydim
){
    int i,xx,yy,count_template,count_suspect;
    float line_integral(refined_rotation_dimension),*pli,*pli_template;
    float line_integral_template(refined_rotation_dimension);
    float line_integral_imaginary(refined_rotation_dimension);
    float line_integral_imaginary_template(refined_rotation_dimension);
    float angle_x,suspect_x,suspect_y,template_x,template_y;
    float x_template,y_template,x1_suspect,y1_suspect,dx_suspect,dy_suspect;
    float top_x_suspect=(float)(suspect_xdim-1),top_y_suspect=(float)(suspect_ydim-1);
    float top_x_template=(float)(template_xdim-1),top_y_template=(float)(template_ydim-1);
    float a_const,b_const,weak,dc_suspect,dc_template;
    float new_x,new_y,yaxis_x,yaxis_y,xaxis_x,xaxis_y;
    yaxis_x = (x[2]-x[0])/(float)(suspect_ydim-1); /* this gives the unit vector in terms of
    the suspect array */
    yaxis_y = (y[2]-y[0])/(float)(suspect_ydim-1);
    xaxis_x = (x[1]-x[0])/(float)(suspect_xdim-1);
    xaxis_y = (y[1]-y[0])/(float)(suspect_xdim-1);
    /* create line integral sweep around suspect's and template's center point */
    pli = line_integral; integral_template;
    pli_template = line_integral_template;
    dc_suspect = dc_template=(float)0.0;
    for(i=0;i<REFINED_ROTATION_DIMENSION;i++){
        angle = (float)PI / (float)REFINED_ROTATION_DIMENSION;
        x_suspect = x1_suspect = (float)0.5 + top_x_suspect/(float)2.0;
        y_suspect = y1_suspect = (float)0.5 + top_y_suspect/(float)2.0;
        dx_suspect = (float)sin((double)angle);
        dy_suspect = (float)cos((double)angle);
        x_suspect+=dx_suspect; x1_suspect-=dx_suspect;
        y_suspect+=dy_suspect; y1_suspect-=dy_suspect;
        x_template = x1_template = (float)0.5+x[4];
        y_template = y1_template = (float)0.5+y[4];
        dx_template = (xaxis_x*dx_suspect+yaxis_y*dy_suspect);
        dy_template = (xaxis_y*dx_suspect-yaxis_y*dy_suspect);
        x_template+=dx_template; x1_template-=dx_template;
        y_template+=dy_template; y1_template-=dy_template;
        *pli = (float)0.0;
        *pli_template = (float)0.0;
        count_template=0; count_suspect=0;
        while(x_suspect>0.0 && x_suspect<top_x_suspect && y_suspect>0.0 &&
        y_suspect<top_y_suspect){
            xx = (int)x_suspect;
            yy = (int)y_suspect;
            *pli += suspect[yy*suspect_xdim+xx];
            xx = (int)x1_suspect;
            yy = (int)y1_suspect;
            *pli += suspect[yy*suspect_xdim+xx];
            x_suspect+=dx_suspect; x1_suspect-=dx_suspect;
            y_suspect+=dy_suspect; y1_suspect-=dy_suspect;
            count_suspect++;
        }
        if(y_template>0.0 && y_template<top_y_template && x_template>0.0 && x_template<top_x_template){
            xx = (int)x_template;
            yy = (int)y_template;
            *pli_template += template[yy*template_xdim+xx];
            xx = (int)x1_template;
            yy = (int)y1_template;
            *pli_template += template[yy*template_xdim+xx];
            x_template+=dx_template; x1_template-=dx_template;
            y_template+=dy_template; y1_template-=dy_template;
            count_template++;
        }
        *pli /= (float)count_suspect;
        *pli_template /= (float)count_template;
        dc_suspect += *pli++;
        dc_template += *pli_template++;
    }
    /* now one-d fft them and one d gmf */
    memset(line_integral_imaginary, 0, sizeof(float)*REFINED_ROTATION_DIMENSION);

```

```

memset(line_integral_template_imaginary, 0, sizeof(float)*REFINED_ROTATION_DIMENSION);
pli = line_integral;
pli_template = line_integral_template;
dc_suspect /= (float)REFINED_ROTATION_DIMENSION;
for(i=0; i<REFINED_ROTATION_DIMENSION; i++) {
    *pli++ -= dc_suspect;
    *pli_template++ -= dc_template;
}
fft(line_integral, line_integral_imaginary, REFINED_ROTATION_BITS, 0, wr, wi, 1);
fft(line_integral_template, line_integral_template_imaginary, REFINED_ROTATION_BITS, 0, wr, wi, 1);
gmf_id(line_integral, line_integral_imaginary, line_integral_template, line_integral_template_imaginary,
    REFINED_ROTATION_DIMENSION, REFINED_ROTATION_BITS, &tweak);
tweak *= (float)0.5; // slight damping factor
tweak *= -((float)180.0/(float)REFINED_ROTATION_DIMENSION);
/* update xy0 thru xy3 */
a_const = (float)cos((double)tweak * PI / 180.0);
b_const = (float)sin((double)tweak * PI / 180.0);
new_x = a_const*(x[4]-x[0]) - b_const*(y[4]-y[0]);
new_y = b_const*(x[4]-x[0]) + a_const*(y[4]-y[0]);
x[0] = x[4] - new_x;
y[0] = y[4] - new_y;
new_x = a_const*(x[4]-x[1]) - b_const*(y[4]-y[1]);
new_y = b_const*(x[4]-x[1]) + a_const*(y[4]-y[1]);
x[1] = x[4] - new_x;
y[1] = y[4] - new_y;
new_x = a_const*(x[4]-x[2]) - b_const*(y[4]-y[2]);
new_y = b_const*(x[4]-x[2]) + a_const*(y[4]-y[2]);
x[2] = x[4] - new_x;
y[2] = y[4] - new_y;
new_x = a_const*(x[4]-x[3]) - b_const*(y[4]-y[3]);
new_y = b_const*(x[4]-x[3]) + a_const*(y[4]-y[3]);
x[3] = x[4] - new_x;
y[3] = y[4] - new_y;
return(tweak);
}

int Align::fine_tune_x_y(unsigned char *ttemplate,
    int template_xdim,
    int template_ydim,
    unsigned char *suspect,
    int suspect_xdim,
    int suspect_ydim,
    float *x,
    float *y,
    float *rotation)
{
    //int foo=1;
    float refinement;
    //while(foo){
    /* find xscale, ytrans optimal pair */
    refine_axis(ttemplate, template_xdim, template_ydim, suspect, suspect_xdim,
        suspect_ydim, x, y, 0);
    /* find yscale, ytrans optimal pair */
    refine_axis(ttemplate, template_xdim, template_ydim, suspect, suspect_xdim,
        suspect_ydim, x, y, 1);
    /* fine tune rotation */
    refinement = refined_rotation(x, y, suspect, suspect_xdim, suspect_ydim, ttemplate,
        template_xdim, template_ydim);
    // NOTE: SOME CONFUSION ABOUT WHETHER NEXT LINE SHOULD BE -- OR ++
    rotation += refinement;
    m_alignStatus.refinement = refinement;
    return(1);
}

/* subroutine for direct registration */
int get_corners_and_center(
    float *x,
    float *y,
    float rotation,
    float scale,
    float x_trans,
    float y_trans,
    int xdim,
    int ydim,
    int fftdim,
    int downsample)
{

```

```

float a_const, b_const;
/* the center of the suspect array should translate to...
(fftdim*downsample - 1)/2.0 - x_trans*downsample, same on y??? */
/* note that the origin of the downsampled arrays actually is
positioned at (downsample-1)/2, (downsample-1)/2 in the coordinates of the
original arrays */
x_trans = (float)downsample;
y_trans = (float)downsample;
x[4] = (float)(fftdim*downsample - 1)/(float)2.0 + x_trans;
y[4] = (float)(fftdim*downsample - 1)/(float)2.0 + y_trans;
a_const = (float)cos((double)rotation*PI/180.0)/scale;
b_const = (float)sin((double)rotation*PI/180.0)/scale;
x[0] = x[4] - (a_const*(float)(xdim-1) - b_const*(float)(ydim-1))/(float)2.0;
y[0] = y[4] - (b_const*(float)(xdim-1) + a_const*(float)(ydim-1))/(float)2.0;
x[1] = x[4] + (a_const*(float)(xdim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[1] = y[4] + (b_const*(float)(xdim-1) - a_const*(float)(ydim-1))/(float)2.0;
x[2] = x[4] - (a_const*(float)(xdim-1) - b_const*(float)(ydim-1))/(float)2.0;
y[2] = y[4] - (b_const*(float)(xdim-1) + a_const*(float)(ydim-1))/(float)2.0;
x[3] = x[4] + (a_const*(float)(xdim-1) + b_const*(float)(ydim-1))/(float)2.0;
y[3] = y[4] + (b_const*(float)(xdim-1) - a_const*(float)(ydim-1))/(float)2.0;
return(1);
}

int final_image(
    unsigned char *out,
    int outxdim,
    int outydim,
    unsigned char *in,
    int inxdim,
    int inydim,
    float *x,
    float *y,
    int num_channels,
    int option)
{
    unsigned char *pout;
    int i, j, xx, yy;
    float xaxis_x, current_x, fracy, fracy, ftmp, ftmp2, ftmp3, ftmp4;
    float x_start, y_start, scan_x, scan_y, jump_x, jump_y;
    unsigned char *pin;
    if(option == 1) { // clear ttemplate array
        pout=out;
        for(i=0; i<(num_channels*outxdim*outydim); i++)*(pout++)=(unsigned char)0;
    }
    xaxis_x = (x[2]-x[0])/(float)(inydim-1); // this gives the unit vector in terms of the
    yaxis_y = (y[2]-y[0])/(float)(inxdim-1);
    xaxis_dist = (float)sqrt((double)(yaxis_x*yaxis_x+yaxis_y*yaxis_y));
    xaxis_x = (x[1]-x[0])/(float)(inxdim-1);
    xaxis_y = (y[1]-y[0])/(float)(inxdim-1);
    xaxis_dist = (float)sqrt((double)(xaxis_x*xaxis_x+xaxis_y*yaxis_y));
    /* starts is origin dotted with axes */
    x_start = (-x[0] * xaxis_x - y[0] * xaxis_y)/xaxis_dist/xaxis_dist;
    y_start = (-x[0] * xaxis_x - y[0] * xaxis_y)/yaxis_dist/yaxis_dist;
    scan_x = xaxis_x/xaxis_dist/xaxis_dist;
    scan_y = yaxis_y/yaxis_dist/yaxis_dist;
    jump_x = xaxis_x/xaxis_dist/xaxis_dist;
    jump_y = yaxis_y/yaxis_dist/yaxis_dist;
    pout = out;
    for(i=0; i<outydim; i++){
        ii = (float)i;
        current_x = x_start + ii * jump_x;
        current_y = y_start + ii * jump_y;
        if(num_channels==1){
            for(j=0; j<outxdim; j++){
                if(current_x<(float)0.0 || current_x>(float)(inxdim-1) || current_y<(float)0.0
                    || current_y>(float)(inydim-1))
                    if(option == 0) pout++; // this option preserves the rest of template
                else *(pout++) = (unsigned char)0;
            }
        } else {
            xx = (int)current_x;
            yy = (int)current_y;
            fracy = current_x - (float)xx;
            fracy = current_y - (float)yy;

```

```

pin = sin(yy*inxdim + xx);
ftmp = ((float)1.0-fracy)*((float)1.0-fracy)*((float)*(pin++));
ftmp += (fracy*((float)1.0-fracy)*((float)*pin);
pin += (inxdim-1);
ftmp += ((float)1.0-fracy)*fracy*((float)*(pin++));
ftmp += (fracy*fracy*((float)*pin));
// debug lines, use with option -o, then it draws a dashed line around suspect
if (xx == 0 || yy == 0 || yy == 0 || yy == (inydim-2))*pout++=(unsigned
char)0;
else *(pout++) = (unsigned char)ftmp;
//
*(pout++) = (unsigned char)ftmp;
}
current_x += scan_x;
current_y += scan_y;
}
}
else if (num_channels==3){
for (j=0;j<outxdim;j++){
if (current_x<(float)0.0 || current_x>(float)(inxdim-1) || current_y<(float)0.0
|| current_y>(float)(inydim-1)){
// this option preserves the rest of template
if (option == 0)pout+=3;
else *(pout++) = *(pout++) = *(pout++) = (unsigned char)0;
}
else {
xx = (int)current_x;
yy = (int)current_y;
fracy = current_x - (float)xx;
fracy = current_y - (float)yy;
ftmp1 = ((float)1.0 - fracy)*((float)1.0-fracy);
ftmp2 = fracy*((float)1.0-fracy);
ftmp3 = ((float)1.0 - fracy)*fracy;
ftmp4 = fracy*fracy;
pin = sin(3*yy*inxdim + xx);
ftmp = ftmp1*((float)*pin);
pin++;
ftmp += (ftmp2*((float)*pin);
pin += 3*(inxdim-1);
ftmp += (ftmp3*((float)*pin);
pin++;
ftmp += (ftmp4*((float)*pin);
pin++;
pin = sin(3*yy*inxdim + xx+1);
ftmp = ftmp1*((float)*pin);
pin++;
ftmp += (ftmp2*((float)*pin);
pin += 3*(inxdim-1);
ftmp += (ftmp3*((float)*pin);
pin++;
ftmp += (ftmp4*((float)*pin);
pin++;
pin = sin(3*yy*inxdim + xx+2);
ftmp = ftmp1*((float)*pin);
pin++;
ftmp += (ftmp2*((float)*pin);
pin += 3*(inxdim-1);
ftmp += (ftmp3*((float)*pin);
pin++;
ftmp += (ftmp4*((float)*pin);
pin++;
*(pout++) = (unsigned char)ftmp;
pin = sin(3*yy*inxdim + xx+3);
ftmp = ftmp1*((float)*pin);
pin++;
ftmp += (ftmp2*((float)*pin);
pin += 3*(inxdim-1);
ftmp += (ftmp3*((float)*pin);
pin++;
ftmp += (ftmp4*((float)*pin);
pin++;
*(pout++) = (unsigned char)ftmp;
current_x += scan_x;
current_y += scan_y;
}
}
return(1);
}

// new public subliminal grid stuff, Late April 1996
float *subliminal_grid = new float(130*128);
float *mellin_mag_transform = new float(130*128);
int grid_freq_total = 16;
float *grid_phase = new float(grid_freq_total);
int *grid_x = new int(grid_freq_total);
int *grid_y = new int(grid_freq_total);

int log_polar_remap_public(
float *in,
float *out,
int dim)
{
int i, dim2 = dim/2, xx, yy, j;
float *pin, *pout;
double theta, dx, dy, radius (MELLIN_DIMENSION), x, y, fracy, fracy, *pradius;
int n = MELLIN_DIMENSION;
double increment = pow(2.0, 0.025);

load_grid_family(
){
static int done = 0;
// don't change this without checking its effects on the later grid finding routines
// such as resolve_orientation

```

```

double start = sqrt(32.5);
for (i=0; i<n; i++){
radius[i] = start + pow(increment, (double)i);
}
// pre-filter fourier mag data;
// first add 90 degree separated points for 2root2 improvement
for (i=0; i<64; i++){
for (j=0; j<64; j++){ // output into left half of original array
in[63-i+128*j] += in[(i+i)*128+64*j];
}
}
float local_average, *p1, *p2, *p3;
for (i=0; i<64; i++){
pout = sin[64*129+i]; // output into right half of original array
if (i==0)p1 = in;
else p1 = sin[(i-1)*128];
p2 = sin[i*128];
if (i==63)p3 = sin[63*128];
else p3 = sin[(i+1)*128];
// first element
local_average = (*p1 + *(p1+1) + *(p2+1) + *p3)/(float)5.0;
if (*p2 > (float)100.0 * local_average){
*pout = (float)100.0;
}
else if (local_average < SMALL){
*pout = SMALL;
}
else {
*pout = *p2 / local_average;
p1++; p2++; p3++;
pout += 128;
for (j=1; j<63; j++){
local_average = (*p1-1) + *p1 + *(p1+1) + *(p2+1);
local_average /= (float)8.0;
if (*p2 > (float)100.0 * local_average) *pout = (float)100.0;
else if (*p2 < SMALL) *pout = SMALL;
else *pout = *p2 / local_average;
p1++; p2++; p3++;
pout += 128;
}
} // last element
local_average = (*p1 + *(p1-1) + *(p2-1) + *p3)/(float)5.0;
if (*p2 > (float)100.0 * local_average) *pout = (float)100.0;
else if (*p2 < SMALL) *pout = SMALL;
else *pout = *p2 / local_average;
} // copy horizontal row into vertical column for interp porpoises
for (i=1; i<64; i++){ in[64+i] = in[64+i*128];
pout = out;
for (theta=0.0; j=0; j<n; j++){ theta += (PI/((double)n)/2.0) } {
dx = cos(theta);
dy = sin(theta);
pradius = radius;
pout = out[j];
for (i=0; i<n; i++){
x = (double)dim2 + *pradius * dx;
y = (*pradius++) * dy;
xx = (int)x;
yy = (int)y;
fracy = x - (double)xx;
fracy = y - (double)yy;
pin = sin(yy*dim + xx);
*pout = (float) ((1.0-fracy)*(1.0-fracy)* (double)*pin);
*pout += (float) (fracy*(1.0-fracy)* (double)*pin);
pin += (dim-1);
*pout += (float) ((1.0-fracy)*fracy* (double)*pin);
*pout += (float) (fracy*fracy* (double)*pin);
pout += n;
}
}
return(1);
}

```



```

done = 0; // force it for now
if(!done){
    delete [] wr;
    delete [] wi;
    delete [] mag_buffer;
    done = 1;
}

return(1);
}

// specific to hunt for grid
int add_block_magnitude(
    unsigned char *data,
    float *fourier_mag,
    int n, // power of 2 dimension of fourier mag
    float *buffer, // needs to be n*(n+2) in length
    int xbumps,
    int ybumps,
    int bump_size,
    int xdim,
    int original_xdim, // pixel based jump pointer for moving down rows
    int truncated
){
    // load fourier array with bump data
    unsigned char *pdata = data;
    float *pbuffer;
    for(i=0;i<ybumps;i++){
        pbuffer = tbuffer[i*n];
        load_bump_array(
            pbuffer, // floating point bump array to be filled (output)
            pdata, // input pixel data
            xbumps, // number of bumps in this row (not pixels)
            zdim, // number of channels
            bump_size, // pixels per bump
            original_xdim - xbumps*bump_size, // number of raw pixels between
            (xdim*bump_size) and entire image array x dimension
            0 // do not overfill the bump buffer
        );
        pdata += (xdim*original_xdim*bump_size);
    }
    // window it if you please
    float *window_function = new float[128];
    load_window_function(128, window_function);
    float *pwindow_row = window_function;
    float *pwindow_column = window_function;
    pbuffer = buffer;
    for(i=0;i<128;i++){
        pwindow_column = window_function;
        for(j=0;j<128;j++){
            *pbuffer++ = *pwindow_row * *pwindow_column++;
        }
        pwindow_row++;
    }
    delete [] window_function;
    // // this doesn't seem to help at all! results seem to get worse

    // ffr the dog
    int bits = (int) (log( (double)(n+1) ) / log( 2.0 ) ); // ffdim should always be power
    of 2
    realfft2d_in_place(buffer, bits, 0, wr, wi);

    // now add its magnitude into the accumulator array
    float *pfourier = fourier_mag;
    float *preal = buffer;
    float *pimag = tbuffer[n];
    for(i=0;i<(n/2+1);i++){
        for(j=0;j<n;j++){
            // consider a "cheaty" version of the following, just add the absolute mags,
            *pfourier++ = (float)sqrt(*preal * *preal + *pimag * *pimag);
            *preal++ = *preal;
            *pimag++ = *pimag;
        }
    }
    return(1);
}

int rotate_scale_image(
    unsigned char *data,

```

```

int xdim,
int ydim,
int zdim,
int bump_size,
int n,
int original_xdim,
float rotation,
float scale,
float *out
){
    int n2 = n/2;
    float outcenter = (float)(n-1) / (float)2.0;
    float incenter = (float)(xdim-1) / (float)2.0;
    float incenter = (float)(ydim-1) / (float)2.0;

    // create buffer for input data
    float *buffer = new float(xdim*ydim);

    // load buffer array with bump data
    unsigned char *pdata = data;
    float *pbuffer;
    int i;
    for(i=0; i<ydim; i++){
        pbuffer = &buffer(i*n);
        load_bump_array(
            pbuffer, // floating point bump array to be filled (output)
            pdata, // input pixel data
            xdim, // number of bumps in this row (not pixels)
            zdim, // number of channels
            bump_size, // pixels per bump
            original_xdim - xdim*bump_size, // number of raw pixels between (xdim*bump_size) and entire
            image array x dimension
            0 // do not overfill the bump buffer
        );
        pbuffer += (zdim*original_xdim*bump_size);
    }

    // now rotate and scale the input image inside buffer, into the output image
    // use xdim/2 and ydim/2 as the center of rotation for the input image
    // use n/2 and n/2 as the center of the output array
    scale = (float)1.0 / scale;
    rotation = -rotation;
    float costheta = (float)cos( (float)cos( (double) rotation * PI / 180.0 ));
    float sintheta = scale * (float)sin( (double) rotation * PI / 180.0 );
    float ii, jj, fracc, fracy, *pout = out, *pin, x, y;
    for(i=0; i<ydim; i++){
        ii = (float)i - outcenter;
        for(j=0; j<n; j++){
            jj = (float)j - outcenter;
            x = ii * costheta + jj * sintheta;
            y = ii * costheta - jj * sintheta;
            x *= incenter;
            y *= incenter;
            xx = (int)x;
            yy = (int)y;
            if(xx < 0){
                xx = 0;
                fracc = (float)0.0;
            }
            else if(xx >= xdim-1){
                xx = xdim-2;
                fracc = (float)1.0;
            }
            else fracc = x - (float)xx;
            if(yy < 0){
                yy = 0;
                fracy = (float)0.0;
            }
            else if(yy >= ydim-1){
                yy = ydim-2;
                fracy = (float)1.0;
            }
            else fracy = y - (float)yy;

            pin = &buffer[yy*n + x];
            *pout = ( (float)1.0-fracc)*(float)1.0-fracy)* *pin +
                *pout + ( fracc*(float)1.0-fracy)* *pin;
            pin += (n-1);
            *pout += ( (float)1.0-fracc)*fracy* *pin +
                *(pout++) + ( fracc*fracy * *pin );
        }
    }

    delete [] buffer;
}

return(1);
}

/* this is a specialized function simply meant to find out which of 4
90 degree orientations is the true orientation of the subliminal grid;
the fourier mellin transform, combined with our "folding" of frequencies,
gives this ambiguity in the first place
*/
int resolve_orientation(
    unsigned char *data,
    int xdim,
    int ydim,
    int zdim,
    int bump_size,
    int n, // power of 2 used in inverse fft's
    int original_xdim,
    float *rotation,
    float *scale
){
    int mult = 1;
    if(*scale > (float)1.25){ // up n to the next higher power of two
        n*=2;
        mult = 2;
    }

    float *buffer = new float[n*(n+2)];
    int n2 = n/2, i, j;
    rotate_scale_image(
        data,
        xdim,
        ydim,
        zdim,
        bump_size,
        n,
        original_xdim,
        *rotation,
        *scale,
        buffer
    );

    // fft the thing
    int bits = (int) (log( (double)(n+1) ) / log( 2.0 ) ); // fftdim should always be power
    of 2
    realfft2d_in_place(buffer, bits, 0, vr, wi); // ultimately, direct calculation may be faster
    assuming frequency points < bits*bits

    // save the original phase values
    float *real = new float[grid_freq_total];
    float *imag = new float[grid_freq_total];
    for(i=0; i<grid_freq_total; i++){
        real[i] = buffer[n2 + mult*grid_x[i] + 2*n*mult*grid_y[i]];
        imag[i] = -buffer[n + n2 + mult*grid_x[i] + 2*n*mult*grid_y[i]];
    }

    // now step through the four possible orientations, finding the best fit
    // the current incarnation of this routine is intimately tied to
    // the function load_grid_family
    float highest, high = (float)-1e20, grid_real, grid_imag;
    int high_i, tmp;
    float value[4], x_offset[4], y_offset[4];
    for(i=0; i<4; i++){
        // zero out buffer
        memset(buffer, 0, sizeof(float)*n*(n+2));
        // multiply this orientation by saved phases
        for(j=0; j<grid_freq_total; j++){
            if(i==0){
                grid_real = (float)cos((double)grid_phase[j]);
                grid_imag = (float)sin((double)grid_phase[j]);
            }
            else if(i==1){
                tmp = j+grid_freq_total/2;
                grid_real = (float)cos((double)grid_phase[tmp]);
                grid_imag = (float)sin((double)grid_phase[tmp]);
            }
            else if(i==2){
                tmp = j+grid_freq_total/2;
                grid_real = (float)cos((double)grid_phase[tmp]);
                grid_imag = (float)sin((double)grid_phase[tmp]);
            }
            else if(i==3){
                tmp = j+grid_freq_total/2;
                grid_real = (float)cos((double)grid_phase[tmp]);
                grid_imag = (float)sin((double)grid_phase[tmp]);
            }
        }

        buffer[n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]] = real[j] * grid_real -
            imag[j] * grid_imag;
        buffer[n + n2 + mult*grid_x[j] + 2*n*mult*grid_y[j]] = real[j] * grid_imag +
            imag[j] * grid_real;
    }
}

```

```

}
// ultimately, direct calculation may be faster
assuming frequency_points < bits*bits

// find highest point
highest = (float)-1e20;
float *pbuffer = buffer;
int high_x, high_y;
for(j=0; j<(n-n); j++) {
    if( *pbuffer > highest ) {
        highest = *pbuffer;
        high_x = j/n;
        high_y = j - high_x;
    }
    pbuffer++;
}

// load its median inter-sample value
value[i] = get_2d_median(buffer, n, n, high_x, high_y, &x_offset(i), &y_offset(i));

// then, find the highest of the four
if(highest > high) {
    high = i;
    high = highest;
}

// update rotation
*rotation += (float)high * (float)90.0;

delete [] real;
delete [] imag;

return(i);
}

/*
This function performs two basic services: first, it simply attempts
to determine if a public subinimal grid exists or not;
if one does exist, then the second basic service is to determine the
rough scale and rotation state of that grid.

The mode_flag variable provides options for how fast v. thorough the algorithms
are.
*/

int hunt_for_grid(
    unsigned char *data, // input image, unknown signature status
    int xdim, // its full pixel dimension in x
    int ydim, // ditto in y
    int zdim, // number of channels
    int probable_bump_size, // this is a tricky one to start; to best function,
    // we will need to specify or "recommend" some standard bumps-per-inch
    // and first look for the signatures in that region
    int total_blocks, // how hard do we look
    int *present,
    float *scale,
    float *rotation,
    float *mellin_mag_transform
) {
    int xblocks, yblocks, i, j, xlength, ylength;
    unsigned char *pdata;

    // the checking takes the first N 128by128 bump regions, FFT's them,
    // converts them to magnitudes, adds them all, then does
    // the fourier \mellin check between the added versions and
    // the master public grid FM profile.
    // A Yes/No is generated based on the S/N found between a peak and the
    // background

    // find and use full integral blocks only, unless the data is shorter
    // than a full integral block
    int xumpsiz = xdim/probable_bump_size;
    int yumpsiz = ydim/probable_bump_size;
    xblocks = xumpsiz;
    yblocks = yumpsiz / SIGNATURE_BLOCK_DIMENSION; // if 0, doesn't even cover one block but will
    still function

    // temporary
    total_blocks = xblocks * yblocks; // again, 0 will function

    // create the basic fourier magnitude array (SIGDIM*(SIGDIM/2+1)) or 128 by 65
    int n-SIGNATURE_BLOCK_DIMENSION;
    float *fourier_mag = new float[n*(1+n/2)]; // only stores the magnitude
    float *buffet = new float[n*(n-2)]; // give it a full array for processing inside 'add_block'
    int m = MELLIN_DIMENSION;

```

```

scale_buf[0] = (float)pow(increment, (double)scale_buf[0]);
if(xblocks == 0 || yblocks == 0){
    truncated = 1;
    if(xblocks==0)xlength = xbumpsiz;
    else xlength = n;
    if(yblocks==0)ylength = ybumpsiz;
    else ylength = n;
}

// resolve 90 degree ambiguity in rotation/orientation
resolve_orientation(data,xlength,ylength,zdim,probable_bump_size,
n,xdim,rotation_buf[0],scale_buf[0]);

*rotation = rotation_buf[0];
*scale = scale_buf[0];
*present = 1;

//now find precise global alignment parameters
}
else { // send back no go on first detect, then get options for quitting or looking harder
    *present = 0;
}

delete {} rotation_buf;
delete {} scale_buf;
delete {} value;

return(1);
}

int experiment(
    unsigned char *data,
    int n
){
    float *imag = new float[n*n];
    //for(i=0;i<n;i++)imag[i]=(float)0.0;
    load_grid_family(); // will immediately return if already done
    reallfft2d_in_place(subliminal_grid,7,0,wr,wi );
    fft2d(subliminal_grid,imag,7,0,wr,wi );
    return(1);
}

/* main registration program: to be used as main module inside other programs */
int Align::direct_registration {
    unsigned char *ttemplate,
    int template_xdim,
    int template_ydim,
    unsigned char *suspect,
    int suspect_xdim,
    int suspect_ydim,
    int num_channels
    ){
    if(1){
        //experiment(ttemplate,template_xdim);
        //return(1);

        int present;
        float rotation,scale;
        extern float *mellin_mag_transform;
        hunt_for_grid(
            suspect,
            suspect_xdim,
            suspect_ydim,
            num_channels,
            1,
            10,
            *present,
            *scale,
            *rotation,
            mellin_mag_transform
        );
        // temporary: place mellin_mag_transform into ttemplate for return
    }
}

```

```

fftldim, downsamples);
}
memcpy(suspect_copy, suspect_real, array_size*sizeof(float));
/* real-valued 2D FFT both suspect and template into it's half-plane complex self */
realfft2d_in_place(template_real, bits, 0, wr, wi);
realfft2d_in_place(suspect_real, bits, 0, wr, wi);
// calculate fourier mellin transform
fourier_mellin_transform(template_real, ftemp, fftldim, suspect_lp_real);
fourier_mellin_transform(suspect_real, ftemp, fftldim, suspect_lp_real);
/* assuming the inputs are both real only, then real 2D FFT each */
realfft2d_in_place(template_lp_real, lp_bits, 0, wr, wi);
realfft2d_in_place(suspect_lp_real, lp_bits, 0, wr, wi);
/* perform generalized matched filter on the two resulting arrays, outputting some number of
likely candidates, with their associated parameters */
gmf(template_lp_real, suspect_lp_real, lp_sampling, lp_bits, number_candidates,
rotation, scale, value, 0);
// change units on rotation and scale for later stages
for(i=0; i<number_candidates; i++){
rotation[i] *= ((float)180.0 / (float)lp_sampling); // converts to degrees
scale[i] = (float)pow((double)scale_increment, (double)scale[i]); // converts to linear scale
}
/* now we have a series of candidates ( or 1, and we just need to get the rotation
and translation information ) wherein one of them should be
the correct one, this next routine sifts through all candidates, including both
the nominal rotation state and the state 180 degrees rotated from the nominal, and
finds which rotation, scale, and translation gives the highest matched filter
output; which then will be passed to the last fine tuning stage*/
// returns best candidate in first element of rotation, scale, x_trans, y_trans
get_best_candidate(number_candidates, ftemp, fftldim, bits, suspect_copy,
1, (suspect_xdim-1)/downsample, 1, (suspect_ydim-1)/downsample, suspect_xdim,
suspect_ydim, downsamples, rotation, scale, x_trans, y_trans, template_real);
/* convert the scale/rotation/translation parameters of the downsampled arrays
into the x and y positions of the four corners of the suspect array, as projected
onto the template array. Precision in keeping track of the various coordinate systems
translates into final alignments to well better than a single pixel, especially
in light of the subtleties involved with downsampling. The four corners
are labelled 0 through 3 in the arrays x and y, where element 0 is the upper left corner
of the suspect, element 1 is the upper right, element 2 lower left, element 3 lower right.
The master 0,0 origin is placed at the upper left of the template array, while
the centerpoints of the two arrays play a role in rotations. The fifth
point in the x and y arrays is the centerpoint, used just so you don't have to
recalculate it all the time*/
get_corners_and_center(x,y,rotation[0],scale[0],x_trans[0],y_trans[0],
suspect_xdim, suspect_ydim, fftldim, downsamples);
/* now fine tune the result using tricky tricks, see notebook of Nov 28, 1995 */
if(num_channels == 1){
for(i=0; i<100; i++){
fine_tune_x_y(template, template_xdim, template_ydim, suspect_xdim,
suspect_ydim, x,y, rotation);
}
}
else if(num_channels == 3){
fine_tune_x_y(template_lum, template_xdim, template_ydim, suspect_lum, suspect_xdim,
suspect_ydim, x,y, rotation);
}
/* last but not least, create the output image array, with various options */
final_image(template, template_xdim, template_ydim, suspect_xdim,
suspect_ydim, x,y, num_channels, 1); // '1' stands for aligned suspect with black everywhere else
/* Record some results of the alignment process in our status structure */
m_alignStatus.rotation = rotation[0];
m_alignStatus.x_scale = scale[0];
m_alignStatus.y_scale = scale[0];
m_alignStatus.x_trans = x_trans[0];
m_alignStatus.y_trans = y_trans[0];
/* free em all */
delete [] template_real;
delete [] template_lp_real;
delete [] suspect_real;
delete [] suspect_lp_real;
delete [] ftemp;
delete [] suspect_copy;
delete [] suspect_lum;
delete [] template_lum;
}
return(1);
}

```

```

/* shell to at least get the main registration program up and running, tested */
#define NEED_MAIN
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// main()
//
// For Geoff's testing purposes, this main() function was used to
// create a stand-alone program which exercised the alignment
// algorithms. This is #ifdef'd out for the windows version.
//
main( int argc, char *argv[] )
{
int template_xdim, template_ydim, suspect_xdim, suspect_ydim;
char template_filename[80], suspect_filename[80];
FILE *inf;

printf("\nTemplate file name please: ");
scanf("%s", template_filename);
printf("\nX dimension and Y dimension of template file: ");
scanf("%d %d", &template_xdim, &template_ydim);
printf("\nsuspect file name please: ");
scanf("%s", &suspect_filename);
printf("\nX dimension and Y dimension of suspect file: ");
scanf("%d %d", &suspect_xdim, &suspect_ydim);

unsigned char *img1 = new unsigned char[template_xdim*template_ydim*sizeof(unsigned char)];
unsigned char *img2 = new unsigned char[suspect_xdim*suspect_ydim*sizeof(unsigned char)];

/* read in binary data into template */
inf = fopen(template_filename, "rb");
if(!inf) {
fprintf(stderr, "register: can't open %s\n", template_filename);
exit(1);
}
fread(img, sizeof(unsigned char), template_xdim*template_ydim, inf);
fclose(inf);

inf = fopen(suspect_filename, "rb");
if(!inf) {
fprintf(stderr, "register: can't open %s\n", suspect_filename);
exit(1);
}
fread(img2, sizeof(unsigned char), suspect_xdim*suspect_ydim, inf);
fclose(inf);

/* returns registered image inside array 'template' */
direct_registration(img, template_xdim, template_ydim, img1, suspect_xdim, suspect_ydim);

/* write out binary data from template */
inf = fopen("reg_out", "wb");
if(!inf) {
fprintf(stderr, "register: can't open %s\n", "reg_out");
exit(1);
}
fwrite(img, sizeof(unsigned char), template_xdim*template_ydim, inf);
fclose(inf);

/* free and clean up */
delete [] img;
delete [] img1;
return(0);
}

#endif //NEED_MAIN
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// FILE: Align.h
//
// DESCRIPTION:
// Header file for the Alignment core algorithm code and the "Align"
// class used to encapsulate this code.
//
// The Alignment code is equivalent to Geoff Rhoads "Register" core
// algorithms, which were first created and run as a stand-alone C program
// on the SGI, then ported to Win95 and Visual C++ as a "console" program,
// and finally incorporated into the Signer Windows application.
//
// Copyright (C) 1996 Digmarc Incorporated, all rights reserved.
//
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// #ifndef ALIGN_H

```

```

#define ALIGN_H
// A structure used to define results of the alignment process.
typedef struct
{
    float rotation;
    float x_scale;
    float y_scale;
    float x_trans;
    float y_trans;
    float refinement;
} AlignStatus;

// Function prototypes: entry functions
class Align
{
public:
    Align();
    int direct_registration(unsigned char *template,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        int num_channels);
    // Accessor for status
    const AlignStatus GetAlignStatus(void) const {return m_alignStatus;}

private:
    // Private structure which contains results of alignment
    AlignStatus m_alignStatus;

    int fine_tune_x_y(unsigned char *template,
        int template_xdim,
        int template_ydim,
        unsigned char *suspect,
        int suspect_xdim,
        int suspect_ydim,
        float *x,
        float *y,
        float *rotation);
};

// Function prototypes: private functions
int gmf_id(float *real1,
    float *imaginary1,
    float *real2,
    float *imaginary2,
    int dim,
    int bits,
    float *offset);

#endif // ALIGN_H

// AlignDlg.cpp : implementation file
//
#include "stdafx.h"
#include "aligner.h"
#include "AlignDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNAMIC(AlignDlg, CFileDialog)

AlignDlg::AlignDlg(BOOL bopenFileDialog, LPCTSTR lpszDefExt, LPCTSTR lpszFileName,
    DWORD dwFlags, LPCTSTR lpszFilter, CWnd* pParentWnd) :
    CFileDialog(bopenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter, pParentWnd)
{
    BEGIN_MESSAGE_MAP(AlignDlg, CFileDialog)
        //{{AFX_MSG_MAP(AlignDlg)
        // NOTE - the ClassWizard will add and remove mapping macros here.

```

```

// Image may be top to bottom or bottom to top.
// We must generate snow accordingly
if (bmiHeader->biHeight > 0)
{
    bottom_up = TRUE;
    line = bmiHeader->biHeight - 1;
}
else
{
    bottom_up = FALSE;
    line = -0;
}

// Generate snow one image scan line at a time.
for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
{
    // Set pointer to first byte for this scan line.
    p_line = &image_data[line * (long) width_in_bytes];
    for (i = 0, j = 0; i < bmiHeader->biWidth; i++)
    {
        if (bmiHeader->biBitCount == 24)
        {
            // For 24 bit color case, need r,g,b snow...
            p_line[j++] = (char) rand();
            p_line[j++] = (char) rand();
            p_line[j++] = (char) rand();
        }
        else
        {
            // For test to make grey-scale and color keys match
            // we must call rand 3 times, but only keep same value
            // as the green channel of the rgb version. This way,
            // if we convert color image to greyscale we can read it.
            p_line[i] = (char) rand(); // we make grey snow same as green.
            rand();
            rand();
        }
    }
    if (bottom_up) line--;
    else line++;
}

void CoXKey::UseNewKey(unsigned newkey)
{
    char *line;
    int width_in_bytes, line_cnt, i;

    // Save the new key.
    user_key = newkey;

    width_in_bytes = (int) WIDTHBYTES(bmiHeader->biWidth * bmiHeader->biBitCount);
    // Seed the random number generator
    srand(user_key);

    for (line_cnt = 0; line_cnt < bmiHeader->biHeight; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        line = &image_data[line_cnt * (long) width_in_bytes];
        for (i = 0; i < bmiHeader->biWidth; i++)
        {
            line[i] = (char) rand();
        }
    }
}

//*****
// FILE: CoXKey.h
//*****
// DESCRIPTION:
// The CoXKey (for Coextensive Key) class encapsulates the functions and
// data structures used to generate a "snowy image" of the same extent
// (i.e., x, y dimensions) as the input image.
// This header file should be included by any module which creates or
// makes use of CoXKey objects.
// CREATION DATE: August 15, 1995
// Copyright (c) 1995 Digimarc Incorporated, all rights reserved.
//*****

#endif COXKEY_H
#define COXKEY_H

#include "digimarc.h"
#include "Params.h"
#include "RawImage.h"
#include "stdafx.h"
#include "afx.h"

class CoXKey
{
public:
    // Public member functions
    // The constructor is passed the user key value and ptrs to the DIB header
    // structures and the data space. The header is assumed to be filled out
    // correctly, while the data space is allocated but empty.
    // Alternative: pass an HDIB handle, allowing this class to handle locking.
    // FOR NOW, I ALSO ASSUME THE PALETTE HAS BEEN SET UP (its the same as image we are
    // signing)
    CoXKey(int user_key, HDIB hDib);
    CoXKey(unsigned user_key, BITMAPINFO *bmi, LPSTR lpDIBbits);

private:
    // Private member functions
    // This function may be a useful idea for future, but it needs rework.
    // I'm making it private to assure no one is calling it.
    void UseNewKey(unsigned newkey);

private:
    // Private data
    // Copy of the user key value.
    unsigned user_key;
    // Pointers to the bitmap info header structure, and the palette array.
    BITMAPINFOHEADER *bmiHeader; // Points to header structure
    RGBQUAD *bmiColors; // Pts to beginning of palette array
    LPSTR lpDIBbits; // Pointer to DIB bits
    char *image_data; // Pointer to raw image data.
};

#endif COXKEY_H

//*****
// dibapi.cpp
//*****
// Source file for Device-Independent Bitmap (DIB) API. Provides
// the following functions:
// PaintDIB() - Painting routine for a DIB
// CreateDIBPalette() - Creates a palette from a DIB
// FindDIBbits() - Returns a pointer to the DIB bits
// DIBWidth() - Gets the width of the DIB
// DIBHeight() - Gets the height of the DIB
// PaletteSize() - Gets the size required to store the DIB's palette
// DIBNumColors() - Calculates the number of colors
// CopyHandle() - Makes a copy of the given global memory block
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.
#include "stdafx.h"
#include "dibapi.h"
#include "io.h"
#include "errno.h"
//*****
// PaintDIB()
// Parameters:
//*****

```

```

* hDC hDC
* LPRECT lpDcRect - rectangle on DC to do output to
* HDIB hDIB - handle to global memory with a DIB spec
* LPRECT lpDIBRect - rectangle of DIB to output into lpDcRect
* CPalette* pPal - pointer to Cpalette containing DIB's palette
* Return Value:
* BOOL - TRUE if DIB was drawn, FALSE otherwise
* Description:
* Painting routine for a DIB. Calls StretchDIBits() or
* SetDIBitsToDevice() to paint the DIB. The DIB is
* output to the specified DC, at the coordinates given
* in lpDcRect. The area of the DIB to be output is
* given by lpDIBRect.
*.....*/
BOOL WINAPI PaintDIB(HDC hDC,
LPRECT lpDcRect,
HDIB hDIB,
LPRECT lpDIBRect,
CPalette* pPal)
{
LPSTR lpDIBHdr; // Pointer to BITMAPINFOHEADER
LPSTR lpDIBBits; // Pointer to DIB bits
BOOL bSuccess=FALSE; // Success/fail flag
HPALETTE hPal=NULL; // Our DIB's palette
HPALETTE holdPal=NULL; // Previous palette
/* Check for valid DIB handle */
if (hDIB == NULL)
return FALSE;
/* Lock down the DIB, and get a pointer to the beginning of the bit
* buffer
*/
lpDIBHdr = (LPSTR)::GlobalLock((HGLOBAL) hDIB);
lpDIBBits = ::FindDIBBits(lpDIBHdr);
/* Get the DIB's palette, then select it into DC
if (pPal != NULL)
hPal = (HPALETTE) pPal->m_hObject;
// Select as background since we have
// already realized in foreground if needed
holdPal = ::SelectPalette(hDC, hPal, TRUE);
*/
/* Make sure to use the stretching mode best for color pictures */
::SetStretchMode(hDC, COLORONCOLOR);
/* Determine whether to call StretchDIBits() or SetDIBitsToDevice() */
if ((RECTWIDTH(lpDcRect) == RECTWIDTH(lpDIBRect)) &&
(RECTHEIGHT(lpDcRect) == RECTHEIGHT(lpDIBRect)))
bSuccess = ::SetDIBitsToDevice(hDC,
lpDcRect->left,
lpDcRect->top,
RECTWIDTH(lpDcRect),
RECTHEIGHT(lpDcRect),
lpDIBRect->left,
lpDIBRect->top,
(int)DIBHeight(lpDIBHdr) -
lpDIBRect->stop -
RECTHEIGHT(lpDIBRect),
0,
lpDIBBits,
lpDIBHdr,
lpDIBBits,
lpDIBMAPINFO.lpDIBHdr,
DIB_RGB_COLORS);
else
bSuccess = ::StretchDIBits(hDC,
lpDcRect->left,
lpDcRect->top,
RECTWIDTH(lpDcRect),
RECTHEIGHT(lpDcRect),
lpDIBRect->left,
lpDIBRect->top,
RECTWIDTH(lpDIBRect),
RECTHEIGHT(lpDIBRect),
lpDIBBits,
lpDIBMAPINFO.lpDIBHdr,
DIB_RGB_COLORS,
SRCCOPY);
}

```

```

::GlobalUnlock((HGLOBAL) hDIB);
/* Reselect old palette */
if (holdPal != NULL)
{
::SelectPalette(hDC, holdPal, TRUE);
}
return bSuccess;
}
/*.....*/
* CreatedDIBPalette()
* Parameter:
* HDIB hDIB - specifies the DIB
* Return Value:
* HPALETTE - specifies the palette
* Description:
* This function creates a palette from a DIB by allocating memory for the
* logical palette, reading and storing the colors from the DIB's color table
* into the logical palette, creating a palette from this logical palette,
* and then returning the palette's handle. This allows the DIB to be
* displayed using the best possible colors (important for DIBs with 256 or
* more colors).
*.....*/
BOOL WINAPI CreatedDIBPalette(HDIB hDIB, CPalette* pPal)
{
LPLOGPALETTE lpPal; // pointer to a logical palette
HPALETTE hLogPal; // handle to a logical palette
int i; // handle to a palette
WORD wNumColors; // loop index
LPSTR lpDIB; // number of colors in color table
LPBITMAPINFO lpBmi; // pointer to packed-DIB
LPBITMAPCOREINFO lpBmc; // pointer to BITMAPINFO structure (Win3.0)
BOOL bMinStyleDIB; // pointer to BITMAPCOREINFO structure (old)
BOOL bResult = FALSE; // flag which signifies whether this is a Win3.0 DIB
/* if handle to DIB is invalid, return FALSE */
if (hDIB == NULL)
return FALSE;
lpbi = (LPSTR)::GlobalLock((HGLOBAL) hDIB);
/* get pointer to BITMAPINFO (Win 3.0) */
lpbmi = (LPBITMAPINFO)lpbi;
/* get pointer to BITMAPCOREINFO (old 1.x) */
lpbmc = (LPBITMAPCOREINFO)lpbi;
/* get the number of colors in the DIB */
wNumColors = ::DIBNumColors(lpbi);
if (wNumColors != 0)
{
/* allocate memory block for logical palette */
hLogPal = ::GlobalAlloc(GHND, sizeof(LOGPALETTE)
+ sizeof(PALETTEENTRY)
* wNumColors);
/* if not enough memory, clean up and return NULL */
if (hLogPal == 0)
{
::GlobalUnlock((HGLOBAL) hDIB);
return FALSE;
}
lpPal = (LPLOGPALETTE)::GlobalLock((HGLOBAL) hLogPal);
/* set version and number of palette entries */
lpPal->palVersion = PALVERSION;
lpPal->palNumEntries = (WORD)wNumColors;
/* is this a Win 3.0 DIB? */
bMinStyleDIB = IS_WIN30_DIB(lpbi);
for (i = 0; i < (int)wNumColors; i++)
{
if (bMinStyleDIB)
{

```



```

lpPal->palPalEntry[i].peRed = lpBmi->bmiColors[i].rgbRed;
lpPal->palPalEntry[i].peGreen = lpBmi->bmiColors[i].rgbGreen;
lpPal->palPalEntry[i].peBlue = lpBmi->bmiColors[i].rgbBlue;
lpPal->palPalEntry[i].peFlags = 0;
}
else
{
    lpPal->palPalEntry[i].peRed = lpBmc->bmiColors[i].rgbRed;
    lpPal->palPalEntry[i].peGreen = lpBmc->bmiColors[i].rgbGreen;
    lpPal->palPalEntry[i].peBlue = lpBmc->bmiColors[i].rgbBlue;
    lpPal->palPalEntry[i].peFlags = 0;
}
}

/* create the palette and get handle to it */
bResult = ppal->CreatePalette(lpPal);
::GlobalUnlock((HGLOBAL) hLogPal);
}
::GlobalFree((HGLOBAL) hLogPal);

::GlobalUnlock((HGLOBAL) hDIB);

return bResult;
}

FindDIBBits()
{
    Parameter:
    LPSTR lpbi - pointer to packed-DIB memory block
    Return Value:
    LPSTR - pointer to the DIB bits
    Description:
    This function calculates the address of the DIB's bits and returns a
    pointer to the DIB bits.
}

LPSTR WINAPI FindDIBBits(LPSTR lpbi)
{
    return (lpbi + ((LPDWORD)lpbi * ::PaletteSize(lpbi)));
}

DIBWidth()
{
    Parameter:
    LPSTR lpbi - pointer to packed-DIB memory block
    Return Value:
    DWORD - width of the DIB
    Description:
    This function gets the width of the DIB from the BITMAPINFOHEADER
    width field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
    width field if it is an other-style DIB.
}

DWORD WINAPI DIBWidth(LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi; // pointer to a Win 3.0-style DIB
    LPBITMAPCOREHEADER lpbmc; // pointer to an other-style DIB

    /* point to the header (whether Win 3.0 and old) */
    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    /* return the DIB width if it is a Win 3.0 DIB */
    if (IS_WIN30_DIB(lpbi))
        return lpbmi->biWidth;
    else /* it is an other-style DIB, so return its width */
        return (DWORD)lpbmc->bcWidth;
}

DIBHeight()
{
    Parameter:
    LPSTR lpbi - pointer to packed-DIB memory block
    Return Value:
    DWORD - height of the DIB
    Description:
    This function gets the height of the DIB from the BITMAPINFOHEADER
    height field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
    height field if it is an other-style DIB.
}

DWORD WINAPI DIBHeight(LPSTR lpDIB)
{
    LPBITMAPINFOHEADER lpbmi; // pointer to a Win 3.0-style DIB
    LPBITMAPCOREHEADER lpbmc; // pointer to an other-style DIB

    /* point to the header (whether old or Win 3.0) */
    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    /* return the DIB height if it is a Win 3.0 DIB */
    if (IS_WIN30_DIB(lpbi))
        return lpbmi->biHeight;
    else /* it is an other-style DIB, so return its height */
        return (DWORD)lpbmc->bcHeight;
}

PaletteSize()
{
    Parameter:
    LPSTR lpbi - pointer to packed-DIB memory block
    Return Value:
    WORD - size of the color palette of the DIB
    Description:
    This function gets the size required to store the DIB's palette by
    multiplying the number of colors by the size of an RGBQUAD (for a
    Windows 3.0-style DIB) or by the size of an RGBTRIPLE (for an other-
    style DIB).
}

WORD WINAPI PaletteSize(LPSTR lpbi)
{
    /* calculate the size required by the palette */
    if (IS_WIN30_DIB(lpbi))
        return (WORD)((DIBNumColors(lpbi) * sizeof(RGBQUAD)));
    else
        return (WORD)((DIBNumColors(lpbi) * sizeof(RGBTRIPLE)));
}

DIBNumColors()
{
    Parameter:
    LPSTR lpbi - pointer to packed-DIB memory block
    Return Value:
    WORD - number of colors in the color table
    Description:
    This function calculates the number of colors in the DIB's color table
    by finding the bits per pixel for the DIB (whether Win3.0 or other-style
    DIB). If bits per pixel is 1: colors=2, if 4: colors=16, if 8: colors=256,
    if 24, no colors in color table.
}

```

```

// Params: h == Handle to global memory to duplicate.
// Returns: Handle to new global memory block.
//-----
HANDLE WINAPI CopyHandle (HANDLE h)
{
    BYTE *lpCopy;
    BYTE *lp;
    HANDLE hCopy;
    DWORD dwLen;
    if (h == NULL)
        return NULL;
    dwLen = ::GlobalSize((HGLOBAL) h);
    if ((hCopy = (HANDLE) ::GlobalAlloc (GHND, dwLen)) != NULL)
    {
        lpCopy = (BYTE *) ::GlobalLock((HGLOBAL) hCopy);
        lp = (BYTE *) ::GlobalLock((HGLOBAL) h);
        while (dwLen-- > 0)
            *lpCopy++ = *lp++;
        ::GlobalUnlock((HGLOBAL) hCopy);
        ::GlobalUnlock((HGLOBAL) h);
    }
    return hCopy;
}

// dibapi.h

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or Winhelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#ifndef _INC_DIBAPI
#define _INC_DIBAPI

/* Handle to a DIB */
DECLARE_HANDLE(HDIB);

/* DIB constants */
#define PALVERSION 0x300

/* DIB Macros */

#define IS_WIN30_DIB(lpbi) ((LPDWORD)(lpbi) == sizeof(BITMAPINFOHEADER))
#define RECTWIDTH(lprect) ((lprect)->right - (lprect)->left)
#define RECTHEIGHT(lprect) ((lprect)->bottom - (lprect)->top)

/* WIDTHBYTES performs DWORD-aligning of DIB scanlines. The 'bits'
   parameter is the bit count for the scanline (biWidth * biBitCount),
   and this macro returns the number of DWORD-aligned bytes needed
   to hold those bits. */
#define WIDTHBYTES(bits) (((bits) + 31) / 32 * 4)

/* Function prototypes */
BOOL WINAPI PaintDIB (HDC, LPRECT, HDIB, LPRECT, CPalette* pPal);
BOOL WINAPI CreateDIBPalette (HDIB, HDIB, CPalette* pPal);
LPSTR WINAPI FindDIBBits (LPSTR lpbi);
DWORD WINAPI DIBWidth (LPSTR lpDIB);
DWORD WINAPI DIBHeight (LPSTR lpDIB);
WORD WINAPI PaletteSize (LPSTR lpbi);
WORD WINAPI DIBNumColors (LPSTR lpbi);
WORD WINAPI DIBBitCount (LPSTR lpbi);
HANDLE WINAPI CopyHandle (HANDLE h);

BOOL WINAPI SaveDIB (HDIB, HDIB, CFile* pFile);
HDIB WINAPI ReadDIBFile (CFile* pFile);

#endif // _INC_DIBAPI

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <memory.h>

#define MAX_LINEAR_DIMENSION 4096

/* generates ascii lines for t1-n */
main() {
    int i;

    printf("\n\n");
    for(i=0;i<512;i++){
        printf("t%d",i);
        if(i%16) printf("\n");
        else printf("\n\n");
    }
    printf("\n\n");
    for(i=0;i<1024;i++){
        printf("t%d",i);
        if(i%16) printf("\n");
        else printf("\n\n");
    }
    printf("\n\n");
    for(i=0;i<2048;i++){
        printf("t%d",i);
        if(i%16) printf("\n");
        else printf("\n\n");
    }
    printf("\n\n");
    for(i=0;i<4096;i++){
        printf("t%d",i);
        if(i%16) printf("\n");
        else printf("\n\n");
    }
}

static int t10[] = {
    512,256,768,128,640,384,896,64,576,320,832,192,704,488,960,32,
    544,288,800,160,672,416,528,96,608,352,864,224,736,480,992,16,
    528,272,880,144,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
    548,292,804,164,656,400,912,80,592,336,848,208,720,464,976,48,
    560,304,816,176,688,432,944,112,624,368,880,240,752,496,1008,8,
    520,264,776,136,648,392,904,72,584,328,840,200,712,456,968,40,
    552,296,808,168,680,424,936,104,616,360,872,232,744,488,1000,24,
    536,280,792,152,664,408,952,88,600,344,856,216,728,472,984,56,
    568,312,824,184,696,440,960,100,632,376,888,248,760,504,1016,4,
    516,260,772,132,644,388,900,68,580,324,836,196,708,480,996,36,
```

```

static int irvb(int n, int b)
{
    register int r;
    register int i;
    register int nn;
    register int bb;

    bb = b;
    nn = n;

    switch( bb )
    {
        case 1 : return( t1[nn] );
        case 2 : return( t2[nn] );
        case 3 : return( t3[nn] );
        case 4 : return( t4[nn] );
        case 5 : return( t5[nn] );
        case 6 : return( t6[nn] );
        case 7 : return( t7[nn] );
        case 8 : return( t8[nn] );
        case 9 : return( t9[nn] );
        case 10 : return( t10[nn] );
        default:
            {
                r = 0;
                for( i = 0; i < bb; i++ )
                {
                    r = r << 1;
                    r = r | ( nn & 1 );
                    nn = nn >> 1;
                }
                return( r );
            }
    }
}

/* fft() is a routine that calculates the discrete Fourier transform
 * of two arrays taken to be the real and the imaginary parts of an
 * complex array. It returns the transform in the arrays.
 */
void fft(float *ar, float *ai, int nbits, int inv, float *wr, float *wi, int neww)
{
    float *ar; /* the real part of the array */
    float *ai; /* the imag part of the array */
    int nbits; /* log base 2 of the number of elements in the arrays */
    int inv; /* nonzero to indicate the inverse transform */
    float *wr; /* the real part of an array of coefficients */
    float *wi; /* the imag part of an array of coefficients */
    int neww; /* nonzero to indicate the coefficients must be calcd */

    register float *aar;
    register float *aai;
    register float *pr1;
    register float *pi1;
    register float *pr2;
    register float *pi2;
    register float *r1;
    register float *i1;
    register float *r2;
    register float *i2;
    int i;
    register int j;
    int n;
    float fn;
    register int tpin;
    register int n2;
    register int n1;
    int nb;
    int nblock;
    register int nsep;
    register int nsep2;
    int ns;
    register float areal;
    register float aimag;
    register float wreal;
    register float wimag;
    register float *pwr;
    register float *pwi;
    float w;
    aar = ar;
    aai = ai;
    n = 1 << nbits;
    fn = (float) n;
    if( inv == 0 )
    {
        for( i = 0; i < n; i++ )
        {
            aar[i] = aar[i] / fn;
            aai[i] = -aai[i] / fn;
        }
    }
    if( neww != 0 )
    {
        tpin = (float) 6.283186 / fn;
        n2 = n / 2;
        for( nb = 0; nb < n2; nb++ )
        {
            w = tpin * ( (float) irvb( nb, nbbits-1 ) );
            wr[nb] = (float) cos( (double) w );
            wi[nb] = (float) sin( (double) w );
        }
        nblock = 1;
        nsep = n;
        for( ns = 0; ns < nbits; ns++ )
        {
            nsep2 = nsep;
            nsep = nsep / 2;
            pwr = wr;
            pw1 = wi;
            for( nb=0; nb < nblock; nb++, pwr++, pw1++ )
            {
                n1 = nb*nsep2;
                n2 = n1+nsep;
                pr1 = aar[n1];
                pr2 = aar[n2];
                pi1 = aai[n1];
                pi2 = aai[n2];
                p12 = aai[n1];
                wreal = *pwr;
                wimag = *pw1;
                for( j=0; j<nsep; j++ )
                {
                    r1 = *pr1; r2 = *pr2; i1 = *pi1; i2 = *pi2;
                    areal = wreal * r2 - wimag * i2;
                    aimag = wimag * r2 + wreal * i2;
                    *pr2++ = r1 - areal;
                    *pi2++ = i1 - aimag;
                    *pr1++ = r1 + areal;
                    *pi1++ = i1 + aimag;
                }
            }
            nblock = nblock*2;
        }
        for( i = 0; i < n; i++ )
        {
            j = irvb( i, nbits );
            if( i < j )
            {
                areal = aar[i];
                aimag = aai[i];
                aar[i] = aar[j];
                aai[i] = aai[j];
                aar[j] = areal;
                aai[j] = aimag;
            }
            if( inv == 0 ) aai[i] = -aai[i];
        }
        int fft2d(float *ar, float *ai, int nbits, int inv, float *wr, float *wi)
        {
            int i;
            int j;
            int j1;
            int j2;
            int n;
            float xr;
            float xi;
            n = 1 << nbits;
            for( i = 1; i < n; i++ )
            {
                for( j = 0; j < i; j++ )

```

```

    ij = (i<nbits)+j ;
    ji = (j<nbits)+i ;
    xr = ar[ij] ;
    xi = ai[ij] ;
    ar[ij] = ar[ji] ;
    ai[ij] = ai[ji] ;
    ar[ji] = xr ;
    ai[ji] = xi ;
}

fft( kar[0], kai[0], nbits, inv, wr, wi, 1 ) ;
for( i = 1 ; i < n ; i++ )
{
    fft( kar[i<nbits], kai[i<nbits], nbits, inv, wr, wi, 0 ) ;
}

for( i = 1 ; i < n ; i++ )
{
    for( j = 0 ; j < i ; j++ )
    {
        ij = (i<nbits)+j ;
        ji = (j<nbits)+i ;
        xr = ar[ij] ;
        xi = ai[ij] ;
        ar[ij] = ar[ji] ;
        ai[ij] = ai[ji] ;
        ar[ji] = xr ;
        ai[ji] = xi ;
    }

    for( i = 0 ; i < n ; i++ )
    {
        fft( kar[i<nbits], kai[i<nbits], nbits, inv, wr, wi, 0 ) ;
    }

    return(0) ;
}

void realfft_two_arrays(float *array1, float *array2, int nbits, int inv, float *wr, float *wi, int neww)
{
    register int j ;
    register int n ;
    float temp1, half ;
    register float *ptemp1 ;
    register float *ptemp2 ;
    register float *par ;
    register float *pai ;
    register float *pail ;
    register float *ptemp1_1 ;
    register float *ptemp2_1 ;
    n = 1 << nbits ;
    half = n/2 ;
    if( !inv )
    {
        /* sort the results */
        ptemp1 = temp1 ;
        ptemp2 = temp2 ;
        par = array1 ;
        pai = array2 ;
        *ptemp1 = *(par++) ;
        *ptemp2 = *(pai++) ;
        par1 = karay1[n-1] ;
        pail = karay2[n-1] ;
        ptemp1_1 = temp1 ;
        ptemp2_1 = temp2 ;
        for( j=1 ; j<nhalf ; j++ )
        {
            *(ptemp1++) = (float)0.5 * (*par + *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai + *pail) ;
            *(ptemp1++) = (float)0.5 * (*par - *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai - *pail) ;
            par++ ; par1-- ; pai++ ; pail-- ;
        }
        temp1[i] = *par ;
        temp2[i] = *pai ;
        /* now copy the results back into original arrays */
        memcpy(array1, temp1, n*sizeof(float)) ;
        memcpy(array2, temp2, n*sizeof(float)) ;
    }
    else
    {
        /* re-sort results */
        ptemp1 = temp1 ;
        ptemp2 = temp2 ;
        par = array1 ;
        pai = array2 ;
        *ptemp1++ = *par ;
        *ptemp2++ = *pai ;
        for( j=1 ; j<nhalf ; j++ )
        {
            *(ptemp1++) = (float)0.5 * (*par + *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai + *pail) ;
            *(ptemp1++) = (float)0.5 * (*par - *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai - *pail) ;
            par++ ; par1-- ; pai++ ; pail-- ;
        }
        temp1[i] = *par ;
        temp2[i] = *pai ;
        /* now copy the results back into original arrays */
        memcpy(array1, temp1, n*sizeof(float)) ;
        memcpy(array2, temp2, n*sizeof(float)) ;
    }
}

/* this routine requires that the input array have two more rows of n appended, into which the
row will be placed */
int realfft2_in_place(float *ar, int nbits, int inv, float *wr, float *wi)
{
    register int i ;
    register int j ;
    register int ij ;
    register int ji ;
    register int n ;
    register int nhalf ;
    register float xr ;
    register float xi ;
    register float xrl ;
    register float xil ;
    float temp1[MAX_LINEAR_DIMENSION], temp2[MAX_LINEAR_DIMENSION] ;
    register float *ptemp1 ;
    register float *ptemp2 ;
    register float *par ;
    register float *pai ;
    register float *pail ;
    register float *ptemp1_1 ;
    register float *ptemp2_1 ;
    n = 1 << nbits ;
    nhalf = n/2 ;
    if( !inv )
    {
        /* pre-transpose */
        for( i = 1 ; i < n ; i++ )
        {
            for( j = 0 ; j < i ; j++ )
            {
                ij = (i<nbits)+j ;
                ji = (j<nbits)+i ;
                xr = ar[ij] ;
                ar[ij] = ar[ji] ;
                ar[ji] = xr ;
            }
        }
        for( i = 0 ; i < nhalf ; i++ )
        {
            if( i==0 ) fft( kar[0], kai[0], nbits, inv, wr, wi, 1 ) ;
            else fft( kar[n2*i], kai[n2*i+n], nbits, inv, wr, wi, 0 ) ;
        }
        /* sort and pack results */
        ptemp1 = temp1 ;
        ptemp2 = temp2 ;
        par = kar[n2*i] ;
        pai = kar[n2*i+n] ;
        *ptemp1++ = *(par++) ;
        *ptemp2++ = *(pai++) ;
        par1 = kar[n2*i+n] ;
        pail = kar[n2*i+n] ;
        for( j=1 ; j<nhalf ; j++ )
        {
            *(ptemp1++) = (float)0.5 * (*par + *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai + *pail) ;
            *(ptemp1++) = (float)0.5 * (*par - *par1) ;
            *(ptemp2++) = (float)0.5 * (*pai - *pail) ;
            par++ ; par1-- ; pai++ ; pail-- ;
        }
        temp1[i] = *par ;
    }
}

```

```

temp_i[1] = *pai;

/* now copy the results back into original arrays */
memcpy(kar[n2*i], temp_r, n*sizeof(float));
memcpy(kar[n2*i+n], temp_i, n*sizeof(float));
}

/* transpose */
for( i = 2; i < n; i+=2 ) {
    for( j = 0; j < i; j+=2 ) {
        j1 = (i<nbits)+1;
        xr = ar[ij];
        xi = ar[ij+n];
        xrl = ar[ij+1];
        xil = ar[ij+1+n];
        ar[ij] = ar[j1];
        ar[ij+n] = ar[j1+n];
        ar[ij+1] = ar[j1+1];
        ar[ij+1+n] = ar[j1+1+n];
        ar[j1] = xr;
        ar[j1+n] = xi;
        ar[j1+1] = xrl;
        ar[j1+1+n] = xil;
    }
}

/* place nyquist row into n*n row, and zero out their imaginary rows */
memcpy(kar[n*n], kar[n], n*sizeof(float));
memset(kar[n], 0, n*sizeof(float));
memset(kar[n*n+n], 0, n*sizeof(float));

for( i = 0; i < nhalf+1; i++ ) fft( kar[n2*i], kar[n2*i+n], nbits, inv, wr, wi, 0 );

/* finally, shift the arrays in order to simplify external processing */
for( i=0; i<n+2; i++ ) {
    memcpy(temp_r, kar[i*n], nhalf*sizeof(float));
    memcpy(kar[i*n], kar[nhalf+i*n], nhalf*sizeof(float));
    memcpy(kar[nhalf+i*n], temp_r, nhalf*sizeof(float));
}

}

else {
    /* undo format */
    for( i=0; i<(n+2); i++ ) {
        memcpy(temp_r, kar[i*n], (n/2)*sizeof(float));
        memcpy(kar[i*n], kar[n/2+i*n], (n/2)*sizeof(float));
        memcpy(kar[n/2+i*n], temp_r, (n/2)*sizeof(float));
    }

    fft( kar[0], kar[n], nbits, inv, wr, wi, 1 );
    for( i = 1; i < (1+n/2); i++ ) fft( kar[(2*i)*n], kar[(2*i+1)*n], nbits, inv, wr, wi, 0 );
    memcpy(kar[n], kar[n*n], n*sizeof(float));

    /* transpose */
    for( i = 2; i < n; i+=2 ) {
        for( j = 0; j < i; j+=2 ) {
            j1 = (i<nbits)+1;
            xr = ar[ij];
            xi = ar[ij+n];
            xrl = ar[ij+1];
            xil = ar[ij+1+n];
            ar[ij] = ar[j1];
            ar[ij+n] = ar[j1+n];
            ar[ij+1] = ar[j1+1];
            ar[ij+1+n] = ar[j1+1+n];
            ar[j1] = xr;
            ar[j1+n] = xi;
            ar[j1+1] = xrl;
            ar[j1+1+n] = xil;
        }
    }

    /* re-sort results */
    ptemp_r = temp_r;
    ptemp_i = temp_i;
    par = kar[(2*i)*n];
    *ptemp_r++ = *(par++);
    *ptemp_i++ = *(par++);

    pai = kar[(2+i)*n];
    ptemp_r1 = stemp_r[n-1];
    ptemp_i1 = stemp_i[n-1];
    for( j=1; j<(n/2); j++ ) {
        *ptemp_r++ = *(par - *(pai+1));
        *ptemp_r1-- = *(par + *(pai+1));
    }
}

```

FFT.H

```

/*****
* FILE: Fft.h
*
* DESCRIPTION:
* Include file for Geoff's FFT routines. Callers of the FFT functions
* should include this header file to pick up the function prototypes.
*
* Copyright (C) Digimarc Corporation, 1996, all rights reserved.
*****/

void fft(float *ar, /* the real part of the array */
         float *ai, /* the imag part of the array */
         int nbits, /* log base 2 of the number of elements in the arrays */
         float *wr, /* nonzero to indicate the inverse transform */
         float *wi, /* the real part of an array of coefficients */
         float *wi, /* the imag part of an array of coefficients */
         int neww); /* nonzero to indicate the coefficients must be calced */

int fft2d(float *ar, float *ai, int nbits, int inv, float *wr, float *wi);

void realfft_two_arrays(float *array1, float *array2,
                        int nbits, int inv, float *wr, float *wi, int neww);

int realfft2d_in_place(float *ar, int nbits, int inv, float *wr, float *wi);

/*****
* File: Image.cpp
*
* Contains the implementation for the Image class. Image objects
* are used to contain the image data, and provide a more convenient
* set of services related to accessing the image data as well as
* attribute variables describing the image.
*
* Include "Image.h"
* Include "dibapi.h"
* Include "stdafx.h"
*
* Image(HDIB hDIB)
*
* Constructor which creates an Image object, given a handle to
* a DIB which is already in memory.
*
* Image::Image(HDIB hDIB)
* {
*     BITMAPINFO *bmi_info;
*     m_hpackedData = NULL;
*     m_fileOK = TRUE;
*     // its already been opened.
* }
*****/

```

```

m_nDIB = NULL;
m_lpDIB = (LPSTR)::GlobalLock( (HGLOBAL) m_hDIB);
// NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
// WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
// I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
bmi_info = (BITMAPINFO *) m_lpDIB;
// Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
m_lpBmiHeader = &bmi_info->bmiHeader;
m_lpBmiColors = &bmi_info->bmiColors[0];
// Set the pointer to the image data.
m_hpDIBbits = (unsigned char *)::FindDIBbits(m_lpDIB);
m_BitsPerPixel = m_lpBmiHeader->biBitCount;
m_XDim = m_lpBmiHeader->biWidth;
m_YDim = m_lpBmiHeader->biHeight;
m_Compression = m_lpBmiHeader->biCompression;
m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}
// Image(hDIB hDIB)
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
Image::Image(CString filename)
{
    CFile file;
    CFileException fe;
    BITMAPINFO *bmi_info;
    m_hPackedData = NULL;
    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
        m_fileOK = TRUE;
// Try to read the DIB file, catch any exceptions.
TRY
{
    m_hDIB = ::ReadDIBfile(file);
}
CATCH(CFileException, eLoad)
{
    file.Abort();
    MessageBox(NULL, "Error reading the image file", NULL,
        m_hDIB = NULL;
        m_fileOK = FALSE;
    }
    END_CATCH
m_lpDIB = (LPSTR)::GlobalLock( (HGLOBAL) m_hDIB);
// NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
// WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
// I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
bmi_info = (BITMAPINFO *) m_lpDIB;
// Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
m_lpBmiHeader = &bmi_info->bmiHeader;
m_lpBmiColors = &bmi_info->bmiColors[0];
// Set the pointer to the image data.
m_hpDIBbits = (unsigned char *)::FindDIBbits(m_lpDIB);
m_BitsPerPixel = m_lpBmiHeader->biBitCount;
m_XDim = m_lpBmiHeader->biWidth;
m_YDim = m_lpBmiHeader->biHeight;
m_Compression = m_lpBmiHeader->biCompression;
m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}
// ~Image()
// The destructor for the Image class of objects.
// ~Image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB);
    if (m_hPackedData != NULL)
    {
        ::GlobalUnlock( (HGLOBAL) m_hPackedData);
        ::GlobalFree( (HGLOBAL) m_hPackedData);
    }
}
// MakePackedData()
// This function copies the DIB image data into a packed format. This
// is important for two reasons: 1) the DIB formatted data is arranged
// so that each scan line starts on a long word boundary, so there may
// be up to 3 unused bytes at the end of each scan line in the case of
// 8 bit data. This arrangement is inconvenient when passing the image
// data to the core algorithms. Also, 2) if a palette is being used
// (this is the case for all but 24 bit image data), this routine looks
// up the actual image values using the palette and places these values
// in the packed data array. The member variable m_hPackedData is the
// handle to the packed data.
// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
void Image::MakePackedData(void)
{
    unsigned char *hpLine;
    unsigned char *hpData;
    int line_cnt, line, i;
    BOOLEAN bottom_up;
    // Create space and get handle for the packed data of the image.
    m_hPackedData = ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT,
        if (m_hPackedData == 0)
            AfxThrowMemoryException();
    // Lock the packed data global memory (leave locked until destructor),
    m_hPackedData = (unsigned char *)::GlobalLock( (HGLOBAL) m_hPackedData);
    hpData = m_hPackedData;
    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }
    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    bottom_up = FALSE;
    line = 0;
    // Now go through each line and create the packed array.
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpLine = &m_hpDIBbits[line * (long) m_WidthInBytes];
        for (i = 0; i < m_XDim; i++)
        {
            if (m_BitsPerPixel == 24)
                *hpData++ = hpLine[i];
            else
            {
                // For 8 bit (and any other non 24 bit data) we
                // take the image data to be indices into the color
                // table. We look up the actual value. Note we
                // assume gray-scale (i.e., r,g,b triples are all equal -
                // we read the green.
                *hpData++ = m_lpBmiColors[hpLine[i]].rgbGreen;
            }
            if (bottom_up) line--;
            else line++;
        }
    }
}

```

```

// UnpackData()
// This function moves the contents of the packed data array back into
// the DIB data space. This would be used, for example, after one the
// core algorithms have been used to sign the data in the packed array,
// and we want to update the DIB to reflect the changes. Note that this
// requires that we create our own palette, since otherwise we don't know
// that the new data values have corresponding entries in the palette.
// WARNING: CURRENT IMPLEMENTATION ASSUMES 8 BIT GRAY-SCALE IMAGE DATA
void Image::UnpackData(void)
{
    unsigned char *hplLine;
    unsigned char *hpData;
    int line_cnt, line, i;
    BOOLEAN bottom_up;

    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    bottom_up = FALSE;
    line = 0;

    hpData = m_hpPackedData;
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hplLine = &m_hpDIBBits[line * (long) m_WidthInBytes];
        for (i = 0; i < m_XDim; i++)
        {
            hplLine[i] = *hpData++;
        }
        if (bottom_up) line--;
        else line++;
    }

    // Next, we force the palette to be our standard 8 bit gray-scale
    // palette.
    if (m_BitsPerPixel == 8)
    {
        // Set ptr to beginning of palette
        LPRGBQUAD pal = m_lpBmiColors;
        for (i = 0; i < 256; i++)
        {
            pal[i].rgbBlue = pal[i].rgbGreen = pal[i].rgbRed = i;
        }
    }
    else
    {
        MessageBox(NULL, "Can only unpack 8 bit image data", NULL,
            MB_ICONEXCLAMATION | MB_OK);
    }
}

// File: Image.cpp
// Contains the implementation for the Image class. Image objects
// are used to contain the image data, and provide a more convenient
// set of services related to accessing the image data as well as
// attributes/variables describing the image.
// include "image.h"
// include "dibapi.h"
// include "stdafx.h"
// Image(HDIB hDIB)
// Constructor which creates an Image object, given a handle to
// a DIB which is already in memory.

```

```

// Image(HDIB hDIB)
{
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;
    m_fileOK = TRUE; // its already been opened.
    m_hDIB = hDIB;
    m_lpDIB = (LPSTR)::GlobalLock((HGLOBAL) m_hDIB);
    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0]; // will be null for 24 bit
    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);
    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
    m_YDim = m_lpBmiHeader->biHeight;
    m_Compression = m_lpBmiHeader->biCompression;
    m_WidthInBytes = WIDTHBYTES(m_XDim * m_BitsPerPixel);
}

// Image(HDIB hDIB)
// Constructor which creates an Image object, given the name of a DIB
// or BMP file.
Image::Image(CString filename)
{
    CFile file;
    CFileException fe;
    BITMAPINFO *bmi_info;
    m_hpPackedData = NULL;

    if (!file.Open(filename, CFile::modeRead | CFile::shareDenyWrite, &fe))
    {
        CString msg("Error reading image file: ");
        msg += filename;
        MessageBox(NULL, msg, NULL, MB_ICONINFORMATION | MB_OK);
        m_fileOK = FALSE;
    }
    else
    {
        m_fileOK = TRUE;
    }

    // Try to read the DIB file, catch any exceptions.
    TRY
    {
        m_hDIB = ::ReadDIBFile(file);
    }
    CATCH(CFileException, eLoad)
    {
        file.Abort();
        MessageBox(NULL, "Error reading the image file", NULL,
            MB_ICONINFORMATION | MB_OK);
        m_hDIB = NULL;
        m_fileOK = FALSE;
    }
    END_CATCH

    m_lpDIB = (LPSTR)::GlobalLock((HGLOBAL) m_hDIB);
    // NOTE: THE FOLLOWING MEMBER POINTERS ARE ONLY VALID WHILE
    // WE KEEP THE DIB DATA LOCKED IN MEMORY. FOR THIS IMPLEMENTATION,
    // I LEAVE THE DATA LOCKED UNTIL THE OBJECT IS DESTROYED.
    bmi_info = (BITMAPINFO *) m_lpDIB;
    // Set up a pointer to the BITMAPINFOHEADER and RGBQUAD array.
    m_lpBmiHeader = &bmi_info->bmiHeader;
    m_lpBmiColors = &bmi_info->bmiColors[0];
    // Set the pointer to the image data.
    m_hpDIBBits = (unsigned char *) ::FindDIBBits(m_lpDIB);
    m_BitsPerPixel = m_lpBmiHeader->biBitCount;
    m_XDim = m_lpBmiHeader->biWidth;
}

```



```

m_YDim = m_lpBmiHeader->biHeight;
m_Compression = m_lpBmiHeader->biCompression;
m_WidthInBytes = WIDTHBYTES(m_YDim * m_BitsPerPixel);

// -----
// -image()
// -----
// The destructor for the Image class of objects.
// -----
image::~image(void)
{
    ::GlobalUnlock( (HGLOBAL) m_hDIB );

    if (m_hPackedData != NULL)
    {
        ::GlobalUnlock( (HGLOBAL) m_hPackedData );
    }
}

// -----
// MakePackedData()
// -----
// This function copies the DIB image data into a packed format. This
// is important for two reasons: 1) the DIB formatted data is arranged
// so that each scan line starts on a long word boundary, so there may
// be up to 3 unused bytes at the end of each scan line in the case of
// 8 bit data. This arrangement is inconvenient when passing the image
// data to the core algorithms. Also, 2) if a palette is being used
// (this is the case for all but 24 bit image data), this routine looks
// up the actual image values using the palette and places these values
// in the packed data array. The member variable m_hPackedData is the
// handle to the packed data.

// The force_to_1_chan argument is an optional boolean. It defaults
// to FALSE (see function prototype in image.h). If set to TRUE,
// only 1 channel of packed data is created, even if the image is 3
// channels. This is useful when creating snowy images from RGB
// images, since we currently always want 1 channel snowy images.
// -----
void image::MakePackedData(BOOLEAN force_to_1_chan)
{
    unsigned char *hpLine;
    unsigned char *hpData;
    int line_cnt, line, i, j;
    long size;
    BOOLEAN bottom_up;

    // Create space and get handle for the packed data of the image.
    size = m_YDim * m_YDim;
    // For 24 bit true color, we will pack R,G,B values, so triple the size.
    if (m_BitsPerPixel == 24 && force_to_1_chan == FALSE)
        size *= 3;
    m_hPackedData = ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, size);
    if (m_hPackedData == 0)
        AfxThrowMemoryException();

    // Lock the packed data global memory (leave locked until destructor).
    m_hPackedData = (unsigned char *)::GlobalLock( (HGLOBAL) m_hPackedData );

    hpData = m_hPackedData;

    // Image may be top to bottom or bottom to top.
    if (m_lpBmiHeader->biHeight > 0)
    {
        bottom_up = TRUE;
        line = m_YDim - 1;
    }
    else
    {
        bottom_up = FALSE;
        line = 0;
    }

    // TEST CODE
    // For Geoff: don't let it correct for bottom_up
    // bottom_up = FALSE;
    // line = 0;

    hpData = m_hPackedData;
    for (line_cnt = 0; line_cnt < m_YDim; line_cnt++)
    {
        // Set pointer to first byte for this scan line.
        hpLine = &m_hPackedData[line * (long) m_WidthInBytes];
        for (i = 0, j = 0; i < m_YDim; i++)
        {
            if (m_BitsPerPixel == 24)
            {
                hpLine[j+2] = *hpData++; // red
                hpLine[j+1] = *hpData++; // green
                hpLine[j] = *hpData++; // blue
                j += 3;
            }
            else
            {
                hpLine[i] = *hpData++;
            }
            if (bottom_up) line--;
            else line++;
        }
    }

    // Next, we force the palette to be our standard 8 bit grey-scale
    // palette.

```

```

if (m_BitsPerPixel == 8)
{
    // Set ptr to beginning of palette
    LPKQUAD pal = m_lpBmiColors;
    for (i = 0; i < 256; i++)
    {
        pal[i].rgbBlue = pal[i].rgbGreen = pal[i].rgbRed = i;
    }
    else if (m_BitsPerPixel == 24)
    {
        // Don't do any palette work for 24 bit color; there is no palette.
    }
    else
    {
        MessageBox(NULL, "Can only unpack 8 and 24 bit image data", NULL,
            MB_ICONEXCLAMATION | MB_OK);
    }
}

```

IMAGE.H

```

//*****
// FILE: Image.h
//*****
// DESCRIPTION:
// * The image class is used to read .BMP and .DIB image files, and
// * manage an internal representation of them in memory. The goal is
// * to provide a set of service which insulate the caller from having to
// * deal with the specifics of the DIB format. Also, the approach tends
// * to isolate platform specific and file format specific details to this
// * class. For example, adding support for a different type of file
// * format would affect this class, but not the callers.
// * This header file should be included by any module which creates or
// * makes use of Image objects.
// * CREATION DATE: September 5, 1995
// * Copyright (c) 1995 Digimarc Incorporated, all rights reserved.
//*****
#ifndef IMAGE_H
#define IMAGE_H
#include "stdafx.h"
#include "dibapi.h"

```

```

class Image
{
public:
    // Constructors...
    Image(HDIB hDIB); // Takes a handle to a loaded DIB
    Image(CString filename); // Takes a filename
    ~Image(void);
    void Image::MakePackedData(void);
    void Image::UnpackData(BOOLEAN force_to_1_chan = FALSE);
    // Accessors:
    HDIB GetHDIB(void) {return m_hDIB;}
    LPSTR GetLPDIB(void) {return m_lpDIB;}
    BITMAPINFOHEADER *GetBmiHdr(void) {return m_lpBmiHeader;}
    RGBQUAD *GetPalHdr(void) {return m_lpBmiColors;}
    unsigned char *GetPackedData(void) {return m_hpPackedData;}
    int GetBitsPerPixel(void) {return m_BitsPerPixel;}
    WORD GetBitsPerPixel(void) {return m_BitsPerPixel;}
    WORD GetSizeOfPalette(void) {return ::PaletteSize(m_lpDIB);}
    WORD GetSizeOfHeader(void) {return ::DIBNumColors(m_lpDIB);}
    WORD GetNumColors(void) {return ::DIBNumColors(m_lpDIB);}
    LONG GetXDim(void) {return m_XDim;}
    LONG GetYDim(void) {return m_YDim;}
    BOOL GetFileOK(void) {return m_fileOK;}
    // Private member functions
private:
    // Handle to the DIB.
    HDIB m_hDIB;
    LPSTR m_lpDIB; // Pointer to top of DIB, locked in memory
    // Pointers to the bitmap info header structure, and the palette array.

```

```

LPBITMAPINFOHEADER m_lpBmiHeader; // Points to header structure
RGBQUAD FAR* m_lpBmiColors; // Pts to beginning of palette array
unsigned char *m_hpBmiColors; // Pointer to DIB bits
HANDLE m_hPackedData; // Handle for the packed data space
unsigned char *m_hpPackedData; // Pointer to packed copy of data.
LONG m_XDim; // X dimension of image (number of lines)
LONG m_YDim; // Y dimension of image (number of lines)
int m_BitsPerPixel;
LONG m_WidthInBytes;
DWORD m_Compression;
// No. of bytes used in each line of DIB
BOOL m_fileOK;
};
#endif // IMAGE_H

```

MAINFRM.CPP

```

// mainfrm.cpp : Implementation of the ChainFrame class
//*****
#include "stdafx.h"
#include "signer.h"
#include "mainfrm.h"
#ifdef DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
// ChainFrame
IMPLEMENT_DYNAMIC(ChainFrame, CWnd)
BEGIN_MESSAGE_MAP(ChainFrame, CWnd)
//[[AFX_MSG_MAP(ChainFrame)
ON_WM_CREATE()
ON_WM_PALETTECHANGED()
ON_WM_QUEYVNSPALETTE()
//]]AFX_MSG_MAP
END_MESSAGE_MAP()
// arrays of IDs used to initialize control bars
// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
    // same order as in the bitmap 'toolbar.bmp'
    ID_FILE_NEW,
    ID_FILE_OPEN,
    ID_FILE_SAVE,
    ID_SEPARATOR,
    ID_EDIT_CUT,
    ID_EDIT_COPY,
    ID_EDIT_PASTE,
    ID_SEPARATOR,
    ID_FILE_PRINT,
    ID_APP_ABOUT,
};
static UINT BASED_CODE indicators[] =
{
    ID_SEPARATOR, // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCROLL,
};
// ChainFrame construction/destruction
ChainFrame::ChainFrame()
{
    ChainFrame::~ChainFrame()
}
int ChainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)

```



```

PackedMsg::PackedMsg(int msg_length)
{
    int i;

    m_correctBits = 0;

    // Save the length, and allocate space for the ASCII message.
    msg_length = msg_length;
    m_asciiMsg = new char[msg_length+1];

    // Null out the ascii storage
    for (i = 0; i < m_msgLength+1; i++)
        m_asciiMsg[i] = '\0';

    // Allocate space for the packed message. Note there's no NULL termination.
    m_compactMsg = new char[m_msgLength];

    // Allocate space for the MsgBitArray, which will hold one bit of the
    // packed message in each char of an unsigned char array (this is
    // the format that the current code signer needs).
    // Also, we include space for checksum of same length as 1 char.
    m_msgBitArrayLength = (m_msgLength+1) * PACKED_BITS_PER_CHAR;
    m_msgBitArray = new unsigned char[m_msgBitArrayLength];
    m_readerBitArray = new unsigned char[m_msgBitArrayLength];

}

// The Destructor
PackedMsg::~PackedMsg()
{
    delete [] m_asciiMsg;
    delete [] m_compactMsg;
    delete [] m_msgBitArray;
    delete [] m_readerBitArray;
    delete [] m_recoveredAsciiMsg;
}

// PackMessage()
// Converts the ASCII message into an array of "packed" characters (currently 6 bits per packed character) which require a minimum of bandwidth in the Digimarc signed image.
// void PackMsg::PackMessage(void)
{
    int i;
    char ascii_ch;

    for (i = 0; i < m_msgLength; i++)
    {
        ascii_ch = toupper(m_asciiMsg[i]);

        if (ascii_ch >= '0' && ascii_ch <= '9')
            m_compactMsg[i] = zero + (ascii_ch - '0');
        else if (ascii_ch >= 'A' && ascii_ch <= 'Z')
        {
            m_compactMsg[i] = A + (ascii_ch - 'A');
        }
    }

    // Check for special characters and encode them.
    else switch (ascii_ch)
    {
        case ' ':
            m_compactMsg[i] = space;
            break;
        case '.':
            m_compactMsg[i] = period;
            break;
        case ',':
            m_compactMsg[i] = comma;
            break;
        case ':':
            m_compactMsg[i] = colon;
            break;
        case '/':
            m_compactMsg[i] = slash;
            break;
        case '\\':
            m_compactMsg[i] = backslash;
            break;
        default:
            // Warn user that an undefined character was found.
            CString warn_msg;
            warn_msg = "Sorry, but \"";
            warn_msg += CString(ascii_ch);
            warn_msg += "\" is not part of the Digimarc character set.";
            warn_msg += "\nIt will be replaced by a '?'. ";
            MessageBox(NULL, warn_msg, "Warning", MB_ICONINFORMATION | MB_OK);
            break;
    }
}

```



```

* FILE: Params.cpp
*
* DESCRIPTION:
* Implementation of the Parameters classes: SignerParams and
* ReaderParams.
*
*
* CREATION DATE: September 8, 1995
*
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.
* ...../
#include "params.h"
#include "stdafx.h"
#include <string.h>
#include <strstream.h>

//...../
// CONSTRUCTOR FOR SIGNER PARAMS OBJECT WHICH
// TAKES THE COMMAND LINE STRING AS AN ARGUMENT.
//...../

SignerParams::SignerParams(LPSTR cmd_line)
{
    char *dash_ptr, *cmd_type, *cmd, *commands;
    const char *dbg_msg_ptr;

    parameters.input_filename = NULL;
    parameters.message = "Default Message";
    parameters.output_filename = NULL;
    parameters.registry_name = NULL;

    parameters.user_key = 1;
    parameters.gain = (float) 100.0;
    parameters.gamma = (float) 0.07;
    parameters.bump_size = 1;

    parameters.lut_scale = (float) 100.0;
    parameters.super_reader_flag = FALSE;

    dbg_msg_ptr = (const char *) GetMessage();
    TRACE("Debug in SignerParams constructor. Message is: %s\n", dbg_msg_ptr);

    // Make a copy of the command line that we can mutilate
    commands = new char[strlen(cmd_line) + 1];
    strcpy(commands, cmd_line);

    dash_ptr = NULL;

    // If the command line doesn't start w/ a '-', then the command line is
    // a single argument: the filename. This case comes up when the program
    // is invoked by dragging a filename onto the executable in Win95 explorer.
    if (strlen(cmd_line) > 0 && cmd_line[0] != '-')
    {
        parameters.input_filename = new char[strlen(cmd_line) + 1];
        strcpy(parameters.input_filename, cmd_line);
    }
    // Otherwise, we check for the multiple argument format of the command line,
    // in which arguments pairs are used, e.g., "-f <filename>".
    else
    {
        do
        {
            // Find the last '-' character
            dash_ptr = strrchr(cmd_line, '-');
            if (dash_ptr != NULL)
            {
                cmd_type = dash_ptr + 1;
                cmd = cmd_type + 1;

                // Create an in-core input stream
                istrstream inStream(cmd, strlen(cmd));

                switch (*cmd_type)
                {
                    case 'g':
                    case 'G':
                        inStream >> parameters.gain;
                        break;
                    case 'f':
                    case 'F':
                        parameters.input_filename = new char[strlen(cmd) + 1];
                        inStream >> parameters.input_filename;

```

```

                        break;
                    case 'w':
                    case 'W':
                        // parameters.message = new char(strlen(cmd) + 1);
                        // inStream.getline(parameters.message,
                        //                  strlen(cmd)+1,
                        //                  '\0');
                        parameters.message = cmd;
                    case 'z':
                    case 'Z':
                        inStream >> parameters.gamma;
                        break;
                    default:
                        // Lop off the last argument by replacing the dash with a NULL;
                        *dash_ptr = '\0';
                }
                while (dash_ptr != NULL);
            }
            //if (parameters.message == NULL)
            //if (parameters.message = new char(strlen("Default message") + 1);
            //strcpy(parameters.message, "Default message");
            //}

            // Clean up.
            delete [] commands;
        }

        SignerParams::~SignerParams(void)
        {
            if (parameters.input_filename != NULL)
                delete [] parameters.input_filename;

            //if (parameters.message != NULL)
            //delete [] parameters.message;

            if (parameters.output_filename != NULL)
                delete [] parameters.output_filename;

            if (parameters.registry_name != NULL)
                delete [] parameters.registry_name;
        }

        //...../
        // SignerParams::UpdateSignature()
        //...../
        // Update the timestamp member variable within this object.
        //...../
        void SignerParams::UpdateSignature(void)
        {
            // Set the timestamp indicating when we signed this puppy.

            CTime t = CTime::GetCurrentTime();

            parameters.sign_time = t;
        }

        //...../
        // PARAMS.H
        //...../
        *
        * DESCRIPTION:
        * The Params classes are responsible for gathering and managing all
        * user input parameters. There are two classes defined here: 1) the
        * SignerParams class for the signer, and the ReaderParams class for the
        * reader.
        *
        * The SignerParams class also keeps track of internal parameters which
        * control or "tune" the operation of the signer, but which are not
        * accessible by the user.
        *
        * At present, this is a non-GUI version. All
        * user inputs enter from the command line. In the future, a GUI version
        * will be added which will present a dialog box to the user and gather
        * input parameters from a graphical interface. The command line version
        * will probably always exist for testing purposes and possibly batch
        * processing. Different constructors will be used to differentiate
        * between the GUI and cmd line versions.
        *
        * This header file should be included by any module which creates or
        * makes use of SignerParams and/or ReaderParams objects.
        *

```



```

* CREATION DATE: August 15, 1995
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.
* \.....
* #include "stdafx.h"
* #include "digimarc.h"
* #include "time.h"
* #include "stdafx.h"

// Define a structure which will contain the various signer parameters.
// The Signer Params class will contain a private copy of this structure.
typedef struct
{
    // User inputs...
    char *input_filename;
    // User provides some combination of following to uniquely locate
    // the registry entry for the signing event...
    User_Key_t user_key;
    time_t date_of_signing;
    char *registry_name; // optional
    // "Super user" inputs, useful for testing and tuning, go here.

    // Non user inputs will go here...
} reader_param_struct;

class ReaderParams
{
public:
    ReaderParams(int argc, char *argv[]); // Constructor for non-gui (cmd line) version
    // Create an accessor which returns a ptr to a const copy of the parameters structure.
    // An alternative is to write accessors for each individual parameter.
    const reader_param_struct * getParams(void) const;

    // Private member functions and data structures
private:
    reader_param_struct parameters; // structure containing the user parameters.
    // Function which warns user if parameters are not all present or look incorrect.
    // It will also throw an exception if things are not right.
    checkParams(void);
};

#ifdef _PARAMS_H
// TBD: create a Params virtual base class for use by signer and reader params.

class SignerParams
{
public:
    SignerParams(LPSTR cmd_line); // Constructor based on command line
    SignerParams(signer_param_struct *params); // Constructor used during reading, based
    // on reading the registry.
    -SignerParams(void);
    void UpdateSignTime(void);

    // Create an accessor which returns a ptr to a const copy of the parameters structure.
    // An alternative is to write accessors for each individual parameter.
    const signer_param_struct * getParams(void) const;

    // Accessors for specific parameters...
    float GetGain(void) {return parameters.gain;}
    void SetGain(float newgain) {parameters.gain = newgain;}
    float GetGamma(void) {return parameters.gamma;}
    void SetGamma(float newgamma) {parameters.gamma = newgamma;}
    char *GetInputFilename(void) {return parameters.input_filename;}
    const CString& GetMessage(void) {return parameters.message;}
    void SetMessage(CString& newstring) {parameters.message = newstring;}
    UINT GetKey(void) {return (UINT) parameters.user_key;}
    void SetKey(UINT newkey) {parameters.user_key = newkey;}
    const CTime& GetTimestamp(void) {return parameters.sign_time;}
    void GetSuperReaderFlag(void) {return parameters.super_reader_flag;}
    void SetSuperReaderFlag(BOOL newflag) {parameters.super_reader_flag = newflag;}
    int GetBumpSize(void) {return parameters.bump_size;}
    void SetBumpSize(int size) {parameters.bump_size = size;}
    float GetLutScale(void) {return parameters.lut_scale;}
    void SetLutScale(float new_scale) {parameters.lut_scale = new_scale;}

    // Private member functions and data structures
private:
    signer_param_struct parameters; // structure containing the user parameters.
    // Function which warns user if parameters are not all present or look incorrect.
    // It will also throw an exception if things are not right.
    checkParams(void);
};

```

```

* and will make use of the public domain software LibTIFF in order*
* to read and write TIFF files.*
* This header file should be included by any module which creates or*
* makes use of RawImage objects.*
* CREATION DATE: August 15, 1995
* Copyright (c) 1995 Digimarc Incorporated, all rights reserved.*
*.....*/
#define RAWIMAGE_H
#define RAWIMAGE_H
#include "digimarc.h"
#include "Params.h"

// Since the exact internal representation may change, use a typedef.
// This will allow a single change to modify all references to the
// raw image data format.
// Also note that in the future we will need several raw image representation.
typedef long *Raw_Data;

class RawImage
{
// Public member functions and data structures
public:
    RawImage(SignerParams *params);

    // Member function which gives caller access to the raw image and its attributes.
    const int getXdin(void);
    const int getYdin(void);

    // This accessor returns a const pointer to a read-only image.
    const Raw_Data getImage(void) const;

    // This accessor returns a const pointer to a writable image.
    Raw_Data * getWritableImage(void) const;

    //Member function used to convert the raw image to an output TIFF file.
    WriteTIFF(char *filename);

// Private data. Users of rawimage objects get at these through accessors only.
private:
    int xdim; // X dimension of image
    int ydim; // Y dimension of image
    Raw_Data image; // Ptr to array of image data
};

#ifdef RAWIMAGE_H

// Read.CPP
// FILE: Read.dpp
// DESCRIPTION:
// Core recognition functions of the Digimarc technology
// Created August 1995
// This particular code uses "raster" based processing as opposed to 2D based
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//.....*/
#include "read.h"
#include "sign.h"
#include "fft.h"
#include "stdafx.h"
#include "math.h"

/* Constants */
const float epsilon = (float) 0.000001;

// read_sbit_single_channel_or_color()
// Used to read (or "recognize") the embedded digimarc signature in
// either a gray-scale or color image. Set number_channels to 1 for
// gray-scale, 3 for color.
//.....*/
int read_sbit_single_channel_or_color(
    unsigned char *data, /* input data to be recognized */
    long original_xdim, /* it's x dimension */

```

```

long original_ydim,
long x_offset,
long y_offset,
long x_extent,
long y_extent,
int message_length,
unsigned char *key,
long key_length,
/* unused */
char *key_lut,
float *luminance_lut,
float *detail_lut,
unsigned char *thumbnail,
unsigned char *original_data,
/* if available, use pointer, otherwise NULL */
const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
float *range, // we will compute a return a crude metric indicating confidence.
unsigned char *message,
int number_channels,
int reading_mode,
int bumps
){
    int status = 1;

    if(reading_mode == 0){
        read_8bit_single_channel_OLD_plus_color(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels, bumps);
    }
    else if(reading_mode == 1){
        read_super(
            data, original_xdim, original_ydim, x_offset, y_offset,
            x_extent, y_extent, message_length, key, key_length, key_lut,
            luminance_lut, detail_lut, thumbnail, original_data, referenceBitArray,
            metric, range, message, number_channels, bumps);
    }
    return(status);
}

// read_8bit_single_channel_OLD_plus_color()
//
// void read_8bit_single_channel_OLD_plus_color(
//     unsigned char *data, // input data to be recognized *
//     long original_xdim, // it's x dimension *
//     long x_offset, // x offset of segment *
//     long y_offset, // y offset of segment *
//     long x_extent, // x extent of segment *
//     long y_extent, // y extent of segment *
//     int message_length, // length of message in BITS, also length of message string *
//     unsigned char *key, // original 8 bit random key *
//     long key_length, // key_length often equal to data_length but not always *
//     /* unused */
//     char *key_lut, // look up table mapping key value *
//     float *luminance_lut, // look up table mapping the signature level to luminance *
//     float *detail_lut, // look up table mapping the signature level to luminance *
//     unsigned char *thumbnail, // if available, use pointer, otherwise NULL *
//     unsigned char *original_data, // if available, use pointer, otherwise NULL *
//     const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
//     float *range, // we will compute a return a crude metric indicating confidence.
//     unsigned char *message, // output: either 0 or 1, i.e. inefficient but simple *
//     int number_channels,
//     int bumps
// ){
//     unsigned char *pkey, *pdata;
//     long i, line, bit;
//     int temp, status=1;
//     float *key_value = new float[x_extent];
//     float *data_float = new float[x_extent];
//     float *orig_float = new float[x_extent];
//     float *bit_total = new float[x_extent];
//     //float *bit_mag = new float(message_length);
//     float *pkey_value, *pdata_float;

```

```

float filter_cf = (float)0.5; // kludge for now
double maxdiff = 40.0; // kludge for now

int key_length = 1+(original_xdim-1)/bumps;
for(i=0; i<message_length; i++)
{
    bit_total[i] = (float) 0.0;
    //bit_mag[i] = (float) 0.0;
}
pdata = data;
for(line=y_offset; line<(y_offset+y_extent); line++)
{
    /* FIRST: if either the original image or a thumbnail of the original is available,
    then use either a simple or "advanced" dot product to remove it; "advanced" refers
    to the idea that you may wish to adjust the gamma or higher order stuff */
    //derivative_threshold(data_float, x_extent, number_channels);
    //remove_mean(data_float, x_extent);
    /* load key values */
    int key_offset = (line/bumps)*key_xlength;
    pkey = &key[key_offset + x_offset/bumps];
    pkey_value = key_value;
    if(bumps>1){
        for(i=x_offset; i<(x_offset+x_extent); i++){
            *pkey_value++ = (float){ (int)key_lut[ (int)*pkey ] };
        }
    }
    else {
        for(i=x_offset; i<(x_offset+x_extent); i++){
            *pkey_value++ = (float){ (int)key_lut[ (int)*pkey++ ] };
        }
    }
    pdata+=(number_channels*x_extent);
}

/* now step through processed patch and perform simple or "advanced" correlation
detection, keeping the resultant detection values in the accumulators for each bit of the
message_length
bits */
pdata_float = data_float;
pkey_value = key_value;
float running_average = (float) 0.0;
float ftemp;
for (i = 0; i < MOV_AV_KERNEL; i++)
{
    running_average += *(pdata_float++);
}
float mov_av = (float)MOV_AV_KERNEL;
running_average /= mov_av;
pdata_float = data_float;
temp = MOV_AV_KERNEL/2;
int temp1 = temp+1;
if(bumps>1){
    for (i = x_offset; i < (x_offset + x_extent); i++)
    {
        if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
        else
        {
            ftemp = (*(pdata_float + temp) - *(pdata_float - temp)) / mov_av;
            running_average += ftemp;
        }
        bit = (key_offset + i/bumps) % message_length;
        ftemp = *(pdata_float++) - running_average;
        //bit_mag[bit] += (*pkey_value * *pkey_value);
        bit_total[bit] += (ftemp * *pkey_value++);
    }
}
else {
    for (i = x_offset; i < (x_offset + x_extent); i++)
    {
        if (i <= (x_offset + temp) || i >= (x_offset + x_extent - temp) );
        else
        {
            ftemp = (*(pdata_float + temp) - *(pdata_float - temp)) / (float)
            running_average += ftemp;
        }
        bit = (key_offset + i) % message_length;
        //bit_mag[bit] += (*pkey_value * *pkey_value);
        bit_total[bit] += ( ( *pdata_float++ - running_average ) * *pkey_value++ );
    }
}
// time optimized version of above earlier code
int key_pos = key_offset + x_offset;
for(i=x_offset; i<(x_offset+temp); i++){

```

```

bit = key_foor + message_length;
bit_total[bit] += ( (pdata_float++) - running_average) * (pkey_value++);
}
int temp2 = x_offset + x_extent - temp;
float *pdata_float2 = data_float;
float *pdata_float1 = pdata_float(temp);
for(i=x_offset;temp1;i=temp2;i++){
    running_average += ( (pdata_float1++) - (pdata_float2++) ) / mov_av;
    bit = key_foor + message_length;
    bit_total[bit] += ( (pdata_float++) - running_average) * (pkey_value++);
}
for(i=0;i<temp;i++){
    bit = key_foor + message_length;
    bit_total[bit] += ( (pdata_float++) - running_average) * (pkey_value++);
}
}
}

/* fill the message string based on bit_totals */
for(i=0; i<message_length; i++){
    if(bit_total[i]>0.0)
    {
        message[i]=1;
    }
    else
    {
        message[i]=0;
    }
}

/*
for (i = 0; i < message_length; i++)
{
    // Before normalizing by the magnitudes, be sure we aren't
    // dividing by zero (this happens for an image w/ zero energy.
    if (bit_mag[i] == (float)0.0)
        bit_mag[i] = epsilon;

    bit_total[i] /= (float) sqrt( (double) bit_mag[i] );

}

// Compute the "crude metric", an estimate of rms spread of the
// bit level detector's results. The referenceBitArray is either
// the known message (if it was available to caller) or the
// newly computed estimate of the message.
*metric = get_crude_metric(referenceBitArray, bit_total, range, message_length);

delete [] data_float;
delete [] orig_float;
delete [] bit_total;
delete [] key_value;
delete [] bit_mag;

return;

}

float it()
{
    void float_long_x_extent, int number_channels)
    {
        unsigned char *pdata;
        long i;
        float *pdata;

        pdata = data;
        pdata = data_float;
        if(number_channels == 1){
            for (i = 0; i < x_extent; i++){
                *pdata++ = (float) *pdata++;
            }
        }
        else if (number_channels == 3) {
            for (i = 0; i < x_extent; i++){
                *pdata = (float) *pdata++;
                *pdata += (float) *pdata++;
                *pdata++ = (float) *pdata++;
            }
        }
    }

    int derivative_threshold(float *data, long length, int number_channels, double maxdiff, float
    filter_cf)
    {
        long i;
        int status = 1;
        float *pdata, *last, last;
        double diff;
        float replacement = (float)0.0;
        if(number_channels == 3)maxdiff *= 3.0;
        last = *last = data[0];
        pdata = *data[i];

```

```

for(i=1; i<length; i++){
    diff = (double)*pdata - last;
    last = *pdata;
    if (fabs(diff) > maxdiff){
        if (diff>0.0) diff = replacement;
        else diff = -replacement;
    }
    *pdata = last + (float)diff;
    last = *pdata++;
}

return(status);
}

void read_super(
    unsigned char *data,
    long original_xdim,
    long original_ydim,
    long x_offset,
    long y_offset,
    long x_extent,
    long y_extent,
    long message_length,
    unsigned char *key,
    long key_length,
    /* unused */
    char *key_lut,
    float *luminance_lut,
    float *detail_lut,
    unsigned char *thumbnail,
    unsigned char *original_data,
    /* if available, use pointer, otherwise NULL */
    const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
    float *metric,
    float *range,
    unsigned char *message,
    int number_channels,
    int bumps
){
    /* look up table mapping key value */
    /* look up table mapping the signature level to luminance */
    /* look up table mapping the signature level to luminance */
    /* if available, use pointer, otherwise NULL */
    /* if available, use pointer, otherwise NULL */

    const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
    float *metric,
    float *range,
    unsigned char *message,
    int number_channels,
    int bumps

    /* output: either 0 or 1, i.e. inefficient but simple */

    unsigned char *pkey, *pdata;
    long i, line, bit;
    int status=1, bits, fftdim, j, highest;
    float *bit_total = new float[message_length];
    float *bit_mag = new float[message_length];
    float *key_value = new float[x_extent]*pkey_value;

    int key_length = 1*(original_xdim-1)/bumps;

    for(i=0; i<message_length; i++){
        bit_total[i] = (float) 0.0;
        bit_mag[i] = (float) 0.0;
    }

    // find power of 2 higher than highest dimension
    if(x_extent > y_extent) highest = x_extent;
    else highest = y_extent;
    bits = 1 + (int) (log( (double) highest - 0.5 ) / log(2.0) );
    fftdim = (int) pow(2.0, (double) bits + 0.0000001);

    // create array
    float *image = new float[fftdim*(fftdim+2)];
    float *wr = new float[fftdim];
    float *wi = new float[fftdim];
    float *pimage;
    pimage = image;
    for(i=0; i<fftdim*(fftdim+2); i++){
        *pimage++ = (float) 0.0;
    }

    // convert either a RAW image or a color image to a single floating point luminance image
    float total;
    if(number_channels == 1){
        *pdata = data;
        for(i=0; i<y_extent; i++){
            pimage = image(i*fftdim);
            for(j=0; j<x_extent; j++){
                *pimage = (float) (*pdata++);
                total += *pimage++;
            }
        }
    }
    else if(number_channels == 3){
        *pdata = data;
        for(i=0; i<y_extent; i++){
            pimage = image(i*fftdim);
            for(j=0; j<x_extent; j++){
                if(fabs(pimage[0]) > fabs(pimage[1]) & & fabs(pimage[2]) > fabs(pimage[3]))
                    *pimage++ = (float) fabs(pimage[0]);
                else if(fabs(pimage[1]) > fabs(pimage[2]) & & fabs(pimage[3]) > fabs(pimage[0]))
                    *pimage++ = (float) fabs(pimage[1]);
                else if(fabs(pimage[2]) > fabs(pimage[3]) & & fabs(pimage[0]) > fabs(pimage[1]))
                    *pimage++ = (float) fabs(pimage[2]);
                else
                    *pimage++ = (float) fabs(pimage[3]);
            }
        }
    }
}

```

```

*pmimage = (float) *pdata++;
*pmimage += (float) *pdata++;
*pmimage += (float) *pdata++;
total += *pmimage++;
}

// weird derivative threshold
int ch00=0;
if(ch00){
    // remove dc
    total /= ((float) y_extent * (float) x_extent);
    for(i=0; i<y_extent; i++){
        pimage = image(i*fftdim);
        for(j=0; j<x_extent; j++){
            *pimage++ -= total;
        }
    }

    float *pdetail_vector;
    float *detail_vector = new float[x_extent];
    int start = 5;
    int stop = 500;
    float scale = (float) 0.5;
    for(i=0; i<y_extent; i++){
        get_read_detail_vector(detail_vector, data, x_extent, i, y_extent, number_channels, start, stop, scale,
            image, fftdim);
        pimage = detail_vector;
        for(j=0; j<x_extent; j++){
            *pimage++ += *pdetail_vector++;
        }
        delete [] detail_vector;
    }

    //float filter_cf = (float) 0.5; // kludge for now
    //double maxdiff = 40.0; // kludge for now
    //for(line=0; line<y_extent; line++){
    //    derivative_threshold(image(line*fftdim), x_extent, 1, maxdiff, filter_cf);
    //}

    // easy does the window ??
    // for now, multiply the last four values near the edges by a linear ramp to zero, simply
    // to avoid total edge weirdness
    int window_it=0;
    if(window_it){
        if(x_extent > 10 && y_extent > 10){
            float mult[4], *pmult;
            mult[0] = (float) 0.2; mult[1] = (float) 0.4; mult[2] = (float) 0.6; mult[3] = (float) 0.8;
            *pmult = mult;
            for(i=1; i<5; i++){
                pimage = image((i-1)*fftdim);
                for(j=0; j<x_extent; j++){
                    *pimage++ *= *pmult++;
                }
            }
            *pmult = mult;
            for(i=1; i<5; i++){
                pimage = image((i-1)*fftdim);
                for(j=0; j<x_extent; j++){
                    *pimage++ *= *pmult++;
                }
            }
        }
    }

    // fft arrays
    realfft2d_in_place(image, bits, 0, wr, wi);

    // filter them
    // phase difference only to start
    float mag1, preal1, *pimaginary1;
    // double power = 0.8;
    preal1 = image; pimaginary1 = image(fftdim);
    for(i=0; i<(1+fftdim/2); i++){
        for(j=0; j<fftdim; j++){
            mag1 = (float) fabs( (double) preal1 + (float) fabs( (double) *pimaginary1 ) );
            if(mag1 == (float) 0.0){
                *preal1++ = (float) 0.0;
            }
        }
    }
}

```



```

#ifdef READ_H
#define READ_H

#define SECOND_THRESHOLD (float) 20.0
#define FIRST_THRESHOLD (float) 20.0

#define MOV_AV_KERNEL 5

int derivative_threshold(float *data, long length, int number_channels, double maxdiff,
//void float_it(unsigned char *data, float *data_float, long x_extent);
void float_it(unsigned char *data, float *data_float,
long x_extent, int number_channels);
void remove_mean(float *array, long length);
float get_crude_metric(const unsigned char *actual_message,
float *bit_total,
float *range,
int message_length);

int read_8bit_single_channel_or_color(
unsigned char *data, // input data to be recognized */
long original_xdim, // it's x dimension */
long original_ydim, // it's y dimension */
long x_offset, // x offset of segment */
long y_offset, // y offset of segment */
long x_extent, // x extent of segment */
long y_extent, // y extent of segment */
int message_length, // length of message in BITS, also length of message string */
unsigned char *key, // original 8 bit random key */
long key_length, // key_length often equal to data_length but not always */
/*unused*/
char *key_lut, // look up table mapping key value */
float *luminance_lut, // look up table mapping the signature level to luminance*/
float *detail_lut, // look up table mapping the signature level to detail*/

unsigned char *thumbnail, // if available, use pointer, otherwise NULL*/
unsigned char *original_data, // if available, use pointer, otherwise NULL*/

const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
float *metric, // we will compute a return a crude metric indicating confidence.
float *range, // output: either 0 or 1, i.e. inefficient but simple */
int number_channels, // generally for B&W=1 vs. color == 3
int reading_mode,
int bumps);

void read_8bit_single_channel_OLD_plus_color(
unsigned char *data, // input data to be recognized */
long original_xdim, // it's x dimension */
long original_ydim, // it's y dimension */
long x_offset, // x offset of segment */
long y_offset, // y offset of segment */
long x_extent, // x extent of segment */
long y_extent, // y extent of segment */
int message_length, // length of message in BITS, also length of message string */
unsigned char *key, // original 8 bit random key */
long key_length, // key_length often equal to data_length but not always */
/*unused*/
char *key_lut, // look up table mapping key value */
float *luminance_lut, // look up table mapping the signature level to luminance*/
float *detail_lut, // look up table mapping the signature level to luminance*/

unsigned char *thumbnail, // if available, use pointer, otherwise NULL*/
unsigned char *original_data, // if available, use pointer, otherwise NULL*/

const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
float *metric, // we will compute a return a crude metric indicating confidence.
float *range, // output: either 0 or 1, i.e. inefficient but simple */
int number_channels,
int bumps);

void read_super(
unsigned char *data, // input data to be recognized */
long original_xdim, // it's x dimension */
long original_ydim, // it's y dimension */
long x_offset, // x offset of segment */
long y_offset, // y offset of segment */
long x_extent, // x extent of segment */
long y_extent, // y extent of segment */
int message_length, // length of message in BITS, also length of message string */
unsigned char *key, // original 8 bit random key */
long key_length, // key_length often equal to data_length but not always */
/*unused*/
char *key_lut, // look up table mapping key value */
float *luminance_lut, // look up table mapping the signature level to luminance*/
float *detail_lut, // look up table mapping the signature level to luminance*/

unsigned char *thumbnail, // if available, use pointer, otherwise NULL*/
unsigned char *original_data, // if available, use pointer, otherwise NULL*/

const unsigned char *referenceBitArray, // bit array ptr: either the known message or estimate.
float *metric, // we will compute a return a crude metric indicating confidence.
float *range, // output: either 0 or 1, i.e. inefficient but simple */
int number_channels,
int bumps);

void ReadDig::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(ReadDig)
DDX_Text(pDX, IDC_READ_KEY, m_user_key);
DDV_MinMaxUInt(pDX, m_user_key, 0, 65535);
DDX_Text(pDX, IDC_READ_LENGTH, m_msg_length);
DDV_MinMaxUInt(pDX, m_msg_length, 1, 65535);
DDX_Text(pDX, IDC_READ_GAIN, m_gain);
DDV_MinMaxFloat(pDX, m_gain, 1.e-003f, 1.e+006f);
DDX_Text(pDX, IDC_BUMP_SIZE, m_bump_size);
DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
DDX_Text(pDX, IDC_DETAIL_LUT_SCALIS, m_detail_lut_scale);
DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(ReadDig, CDialog)
//{{AFX_MSG_MAP(ReadDig)
// MESSAGE_MAP
END_MESSAGE_MAP()
}

// Constructor for the Reader Parameters Dialog object. A ReadDig
// object is created to manage a dialog in which the user is able
// to set the parameters used by the Reader and associated core
// algorithms.
// ReadDig::ReadDig(CWnd* pParent /*=NULL*/)
// : CDialog(ReadDig::IDD, pParent)
// {
// {{AFX_DATA_INIT(ReadDig)
// m_user_key = 0;
// m_msg_length = 0;
// m_gain = (float) 0.0;
// m_bump_size = 0;
// m_detail_lut_scale = 0.0f;
// }}AFX_DATA_INIT
// }

// readdig.cpp : implementation file
//
//
#include "stdafx.h"
#include "signer.h"
#include "readdig.h"

#ifdef _DEBUG
#define THIS_FILE static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// ReadDig dialog
//
//
// Constructor for the Reader Parameters Dialog object. A ReadDig
// object is created to manage a dialog in which the user is able
// to set the parameters used by the Reader and associated core
// algorithms.
// ReadDig::ReadDig(CWnd* pParent /*=NULL*/)
// : CDialog(ReadDig::IDD, pParent)
// {
// {{AFX_DATA_INIT(ReadDig)
// m_user_key = 0;
// m_msg_length = 0;
// m_gain = (float) 0.0;
// m_bump_size = 0;
// m_detail_lut_scale = 0.0f;
// }}AFX_DATA_INIT
// }

void ReadDig::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(ReadDig)
DDX_Text(pDX, IDC_READ_KEY, m_user_key);
DDV_MinMaxUInt(pDX, m_user_key, 0, 65535);
DDX_Text(pDX, IDC_READ_LENGTH, m_msg_length);
DDV_MinMaxUInt(pDX, m_msg_length, 1, 65535);
DDX_Text(pDX, IDC_READ_GAIN, m_gain);
DDV_MinMaxFloat(pDX, m_gain, 1.e-003f, 1.e+006f);
DDX_Text(pDX, IDC_BUMP_SIZE, m_bump_size);
DDV_MinMaxInt(pDX, m_bump_size, 1, 256);
DDX_Text(pDX, IDC_DETAIL_LUT_SCALIS, m_detail_lut_scale);
DDV_MinMaxFloat(pDX, m_detail_lut_scale, 1.e-003f, 1.e+006f);
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(ReadDig, CDialog)
//{{AFX_MSG_MAP(ReadDig)
// MESSAGE_MAP
END_MESSAGE_MAP()
}

```

```
////////////////////////////////////
```

```
// ReadDlg message handlers
void ReadDlg::OnOK()
{
    CDialog::OnOK();
}
```

READDLG.H

```
// readdlg.h : header file
//
```

```
////////////////////////////////////
```

```
// ReadDlg dialog
```

```
class ReadDlg : public CDialog
```

```
{
// Construction
```

```
public:
    ReadDlg(CWnd* pParent = NULL); // standard constructor
```

```
// Dialog Data
```

```
    enum { IDD = IDD_READ_DIALOG };
    UINT m_user_key;
```

```
    UINT m_msg_length;
```

```
    float m_gain;
```

```
    int m_bump_size;
```

```
    float m_detail_lut_scale;
```

```
    ///AFX_DATA
```

```
// Implementation
```

```
protected:
```

```
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
```

```
// Generated message map functions
```

```
    ///AFX_MSG(ReadDlg)
```

```
    virtual void OnOK();
```

```
    ///AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
```

```
};
```

RESOURCE.H

```
///([NO_DEPENDENCIES])
```

```
/// Microsoft Developer Studio generated include file.
```

```
/// Used by Signer.rc
```

```
2
3
#define IDR_MAINFRAME
#define IDR_DIBTYPE
#define IDD_ABOUTBOX
#define IDC_MESSAGE
#define IDD_PARAMS_DIALOG
#define IDC_GAIN
#define IDC_GAIN_LABEL
#define IDD_READ_DIALOG
#define IDC_MESSAGE_LABEL
#define IDC_EDIT_GAIN
#define IDC_EDIT_GAMMA
#define IDC_EDIT_KEY
#define IDC_READ_KEY
#define IDC_READ_LENGTH
#define IDC_READ_GAIN
#define IDC_TREE1
#define IDC_BUMP_SIZE
#define IDC_DETAIL_SCALE
#define IDC_DETAIL_LUT_SCALE
#define IDC_EDIT_SETTINGS
#define ID_VIEW_SIGNED
#define ID_VIEW_UNSIGNED
#define ID_VIEW_SNOW
#define ID_VIEW_SNOWY_IMAGE
#define ID_VIEW_STATUS
#define IDC_SETTINGS_SIGNER
#define IDC_SETTINGS_READER
#define IDC_SETTINGS_REGISTRY
#define IDC_SETTINGS_AUTOPRINTREPORT
#define IDC_SETTINGS_AUTOPRINT
#define IDC_SETTINGS_AUTOREAD
#define IDC_SETTINGS_ALIGN
#define IDC_SETTINGS_ALIGN
```

```
/// Next default values for new objects
```

```
///
```

```
#ifndef APSTUDIO_INVOKED
#define APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 106
#define _APS_NEXT_COMMAND_VALUE 32784
#define _APS_NEXT_CONTROL_VALUE 122
#define _APS_NEXT_SYMED_VALUE 102
#endif
```

SIGN.CPP

```
////////////////////////////////////
// FILE: Sign.cpp
//
```

```
// DESCRIPTION:
```

```
// Core signing functions of the digimarc technology.
```

```
// Created July 1995.
```

```
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
```

```
////////////////////////////////////
```

```
#include "sign.h"
```

```
#include <math.h>
```

```
#include "stdafx.h"
```

```
/* this function loads the scaling factor based on luminance */
```

```
int load_luminance_lut( float *luminance_lut, float gamma) // explicitly written for 8 bit
```

```
{
    int i,status=1;
```

```
    luminance_lut[0] = (float) 0.0; /* don't put any signature energy into zero luminance
    (black) */
    for(i=1; i<256; i++)
    {
        luminance_lut[i] = (float) pow((double)i, (double) gamma);
    }
    return(status);
}
```

```
////////////////////////////////////
// load_key_lut()
```

```
// This function just assigns mainly 0's, 1's, -1's, 2's and -2's
// to the key values, scaled by the scale_point, 127
```

```
// scale_point is a simple integer between 1 and 127
```

```
// about 30 to 50 should be about right for first tests
```

```
////////////////////////////////////
```

```
float load_key_lut(char *key_lut, float gain)
```

```
{
    int i,base_gain,ifraction,
    float rms,fraction;
    gain /= (float)100.0;
    rms = gain;
    base_gain = (int)gain;
    ifraction = gain - (float)base_gain;
    ifraction = (int)( (float) 127.0 * ifraction );
    if(ifraction == 0){
        for(i=0; i<128; i++)key_lut[i] = (char)base_gain;
        for(i=0; i<128; i++)key_lut[i+128] = -(char)base_gain;
    }
    else {
        for(i=0; i<(128-ifraction); i++){
            key_lut[i] = (char)base_gain;
            key_lut[i+128] = -(char)base_gain;
        }
        for(i=(128-ifraction); i<128; i++){
            key_lut[i] = (char)(base_gain+1);
            key_lut[i+128] = -(char)(base_gain+1);
        }
    }
    return( rms );
}
```

```
////////////////////////////////////
```

```
// The following functions are core algorithms which include
// 1) additional capabilities for signing Color images, and
// 2)
```

```
////////////////////////////////////
```



```

    pkey++;
    if ( ((i/bumps)*key_xlength+j*bumps)<message_length) == (message_length-1) )
    /*
    time to restart message */
    {
        pmessage = message;
        else pmessage++;
    }
}
else if (number_channels == 3) {
    // data_length is assumed to be the number of pixels, not the number of data bytes
    // RGB packing is assumed, in that order, 3 bytes in a row per pixel: R G B
    if (signing_mode == STANDARD) {
        pdata = data;
        p_out = data_out;
        for (i=0; i<ydlim; i++) {
            // load local detail values for this row
            get_detail_vector(detail_vector, pdata, xdim, i, ydim, detail_lut, number_channels);
            pdetail_vector = detail_vector;
            pkey = key + ((i/bumps)*key_xlength);
            pmessage = message + ((i/bumps)*key_xlength);
            for (j=0; j<xdim; j++) {
                lum_change = key_lut((i/bumps)*key_xlength+message_length);
                if (lum_change == 0) {
                    if (lum_change == 0) {
                        memcpy(p_out, pdata, 3*sizeof(unsigned char));
                        pdata++;
                        p_out++;
                        pdetail_vector++;
                    }
                    else {
                        local_gain = (pdetail_vector++) * luminance_lut(*pdata++);
                        if (abs(lum_change) > 1) { // this is the anti-sparkles check
                            if (lum_change > 0) lum_change = 1;
                            else lum_change = -1;
                        }
                        delta = (float)lum_change * local_gain;
                        if (!(*pmessage))
                            delta = -delta; /* invert current snowy image luminance value ... key */
                        for (k=0; k<3; k++) {
                            ftemp = (float)*(pdata++) + delta;
                            if (ftemp > (float)255.0) (p_out++) = (unsigned char)255;
                            else if (ftemp < (float)0.0) (p_out++) = (unsigned char)0;
                            else (p_out++) = (unsigned char)(ftemp*(float)0.5);
                        }
                    }
                }
                if ( ((j+1)*bumps) == 0 ) {
                    pkey++;
                    if ( ((i/bumps)*key_xlength+j*bumps)<message_length) == (message_length-1) )
                    /* time to restart message */
                    {
                        pmessage = message;
                        else pmessage++;
                    }
                }
            }
        }
        return(status);
    }
}

// The following function prototypes correspond to the more
// advanced signing algorithms and color image signing capabilities
// added in February 1996.
//
// int get_detail_vector(float *detail_vector,
//                       unsigned char *data,
//                       int xdim,
//                       int ydim,
//                       int total_rows,
//                       float *luminance_lut,
//                       int number_channels);
//
// int load_detail_lut( float *detail_lut, float scale); // explicitly written for 8 bit
//
// int sign_8bit_single_channel_or_color(
//     unsigned char *data, // input data to be signed
//     long data_length, // it's length
//     long xdim, // it's x dimension
//     long ydim, // it's y dimension
//     unsigned char *message, // either 0 or 1, i.e. inefficient but simple
//     int message_length, // length of message in bits, also length of message string
//     unsigned char *key, // 8 bit random key, uniformly distributed
//     long key_length, // key_length often equal to data_length but not always
//     *unused // look up table mapping key value
//     float *luminance_lut, // look up table mapping the scaling to luminance values
//     float *detail_lut, // look up table mapping the scaling to luminance values
//     int signing_mode, // current options: STANDARD or STRICT_LUMINANCE
//     unsigned char *data_out, // signed output data in same length and format as input
//     int number_channels, // added in late february 1996 to begin work on 3 color 24 bit
//     int bumps // added in March 1996
// );
//
// #endif // SIGN_H

// FILE: SignDoc.cpp
//
// DESCRIPTION:
// Implementation file for the Document class of the Digimarc Signer.
// This defines the implementation of the Document class
// for the Signer. Under the Microsoft Foundation Class (MFC) architecture,
// the Document/View model is the preferred method. This header file
// defines our additions to the generic Document class created by the
// Visual C++ wizards.
//
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//
// #include "stdafx.h"
// #include "signer.h"
// #include <limits.h>
//
// #include "signdoc.h"
// #include "signview.h"
//
// #include "cokey.h"
// #include "image.h"
// #include "sign.h"
// #include "read.h"
// #include "align.h"
//
// #include "parmsdig.h"
// #include "readdig.h"
//
// #include "afxpriv.h"
// #include "afxext.h"
// #include "mainfrm.h"
//
// For the Signer Parameters dialog object
// For the Reader Parameters dialog object

```



```

void CDibDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}

// Makes a snow image, and sets the member variable m_hSnowyDIB, which
// is a DIB handle to the new snow image DIB. The snow image which is
// created is sized based on the parent DIB handle passed in, and it
// has all the same bitmap header and palette stuff.
// CDibDoc::MakeSnow(HDIB hparentDIB)
// CDibDoc::MakeSnow(HDIB hparentDIB)
{
    int cxDIB, cyDIB;
    long num_pixels, num_colors;
    DWORD total_size, image_byte;
    LPSTR lpDIB, lpSnowyDIB; // Pointer to BITMAPINFOHEADER
    LPBITMAPINFOHEADER lpSnowyDIBHdr;
    HPSTR hpsSnowyDIBbits;
    HPSTR src_data, dest_data; // Huge ptrs for copying the image.

    // HDIB hOriginalDIB = GetOriginalHDIB();
    if (hparentDIB == NULL)
        return;

    // Get the size of the parent DIB
    total_size = GlobalSize((HGLOBAL) hparentDIB);

    // Create space for the snow image (on 1st call only).
    if (m_hSnowyDIB == NULL)
    {
        m_hSnowyDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, total_size);
        if (m_hSnowyDIB == 0)
        {
            MessageBox(NULL,
                "Insufficient memory is available for the 'snowy image'",
                "Digimarc Signer Warning",
                MB_ICONINFORMATION | MB_OK);
            return;
        }

        // Lock the two DIBs in memory
        lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) hparentDIB);
        lpSnowyDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hSnowyDIB);

        src_data = (char *) lpDIB;
        dest_data = (char *) lpSnowyDIB;

        // Copy the BITMAPINFOHEADER, palette, and actual image byte data by byte.
        for (image_byte = 0; image_byte < total_size; image_byte++)
        {
            *dest_data++ = *src_data++;
        }

        // For debug: reset the pointers.
        src_data = (char *) lpDIB;
        dest_data = (char *) lpSnowyDIB;
        if (*src_data != *dest_data)
            TRACE("DEBUG: after copy into snowy image, 1st chars aren't equal\n");

        // We are now all done w/ the Parent DIB. Unlock it.
        ::GlobalUnlock((HGLOBAL) hparentDIB);

        // Get ptr to the snowy dib header space.
        lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;

        hpsSnowyDIBbits = ::FindDIBbits(lpSnowyDIB);

        cxDIB = (int) ::DIBWidth(lpSnowyDIB); // X size of DIB
        cyDIB = (int) ::DIBHeight(lpSnowyDIB); // Y size of DIB
        num_pixels = (long) cxDIB * cyDIB;
        num_colors = ::DIBNumColors(lpSnowyDIB);
        if (lpSnowyDIBHdr->biCompression != 0)
        {
            TRACE("Can't cope with compressed image (compression = %d)\n",
                lpSnowyDIBHdr->biCompression);
            ::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
            return;
        }

        TRACE("width = %d, height = %d, num_pixels = %d\n", cxDIB, cyDIB, num_pixels);
        TRACE("num_colors = %d\n", num_colors);
        if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {

```

```

TRACE("At this time, only build snowy image for 8 or 24 bit images\n");
::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
return;
}

if (m_BitsPerPixel == 8 || m_BitsPerPixel == 24)
{
    CoxKey coxkey(m_pParams->GetKey(), (BITMAPINFO *) lpSnowyDIBHdr,
        hpSnowyDIBBits);
}

::GlobalUnlock((HGLOBAL) m_hSnowyDIB);
}

// Sign()
// This is the function which calls upon the core signing algorithms.
// WARNING: CURRENTLY THIS FUNCTION ASSUMES THAT WE ALWAYS ARE SIGNING
// THE "ORIGINAL IMAGE" DIB. THIS MAY BE A BUG.
// First shot at a function which calls the signer core algorithms
void CDibdoc::Sign(void)
{
    long num_pixels, num_colors;
    image_byte;
    HPSTR src_data, dest_data; // Huge ptrs for copying the image.
    float rms;
    int num_channels;

    HDB hOriginalDIB = GetOriginalHDB();
    if (hOriginalDIB == NULL)
        return;

    // Create space for the signed image DIB.
    m_hSignedDIB = (HDB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
    if (m_hSignedDIB == 0)
    {
        MessageBox(NULL,
            "Insufficient memory is available for the signed image",
            "Digitarc Signer Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Create Image objects for the images. Note that this locks them in memory.
    Image snowyImage(m_hSnowyDIB);
    Image unsignedImage(m_hOriginalDIB);

    // This is ugly, but I have to copy the DIB header stuff into the signed DIB
    // before I can create the signed image object.
    dest_data = (char *) ::GlobalLock((HGLOBAL) m_hSignedDIB);

    // We want to copy the BITMAPINFO structure from the unsigned to the signed DIB
    src_data = unsignedImage.GetLpDIB();

    // Copy the BITMAPINFOHEADER and palette to the signed DIB space, byte by byte.
    for (image_byte = 0; image_byte < unsignedImage.GetSizeofHeader(); image_byte++)
    {
        *dest_data++ = *src_data++;
    }

    ::GlobalUnlock((HGLOBAL) m_hSignedDIB);

    // Now create the signedImage object, which will lock the DIB in memory again.
    Image signedImage(m_hSignedDIB);

    // For each, create a "byte-wise" packed data array from the DIB 4-byte packing
    snowyImage.MakePackedData(FORCE_TO_1_CHANNEL); // snowy image always 1 chan
    unsignedImage.MakePackedData();
    signedImage.MakePackedData();

    num_pixels = (long) unsignedImage.GetXDim() * unsignedImage.GetYDim();
    num_colors = unsignedImage.GetNumColors();

    if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {
        TRACE("At this time, only sign 8 and 24 bit images\n");
        return;
    }

    // Create and load the luminance scaling look up table.
float *luminance_lut = new float[256];
::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

// Create and load the key look up table.
char *key_lut = new char[256];
rms = ::load_key_lut(key_lut, m_pParams->GetGain());

long data_length = unsignedImage.GetXDim() * unsignedImage.GetYDim();

// Create a packed msg (will be a user input in future).
delete m_pPackedMsg;
m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());

// Set up some arguments and call the core signer.
int x_dim = unsignedImage.GetXDim();
int y_dim = unsignedImage.GetYDim();

if (unsignedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
else if (unsignedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

// const float lut_scale = (float)1.0; // Later this will be user controlled.
float *detail_lut = new float[DETAIL_TOTAL];
::load_detail_lut(detail_lut, m_pParams->GetLutScale());

::sign_8bit_single_channel_or_color(unsignedImage.GetPackedData(),
    data_length,
    x_dim,
    y_dim,
    m_pPackedMsg->getMsgBitArray(),
    m_pPackedMsg->getMsgBitArrayLength(),
    snowyImage.GetPackedData(),
    data_length,
    key_lut,
    luminance_lut,
    detail_lut,
    STANDARD,
    signedImage.GetPackedData(),
    num_channels,
    m_pParams->GetBumpSize());

delete [] detail_lut;

// Set the timestamp indicating when we signed this puppy.
m_pParams->UpdateSignTime();

delete [] luminance_lut;
delete [] key_lut;

// Now unpack the data in the Image object, back into the standard DIB format
signedImage.UnpackData();
}

// Read()
// The read function is the interface to the core recognition algorithm.
// It sets up the necessary data structures needed by the core routine
// and makes the call.
void CDibdoc::Read(HDB hSignedDIB, BOOL use_super_reader)
{
    long num_pixels, num_colors;
    int num_channels;
    int reading_mode;

    // Create Image objects for the images. Note that this locks them in memory.
    Image snowyImage(m_hSnowyDIB);
    Image signedImage(hSignedDIB);

    // Create a "byte-wise" packed data array from the DIB 4-byte packing
    signedImage.MakePackedData();
    snowyImage.MakePackedData(FORCE_TO_1_CHANNEL); // Snowy images always 1 ch.
    // unsignedImage.MakePackedData();

    num_pixels = (long) signedImage.GetXDim() * signedImage.GetYDim();
    num_colors = signedImage.GetNumColors();

    if (m_BitsPerPixel != 8 && m_BitsPerPixel != 24)
    {

```

```

TRACE("At this time, only recognize 8 and 24 bit images\n");
return;
}

// Create and load the luminance scaling look up table.
float luminance_lut = new float[256];
::load_luminance_lut(luminance_lut, m_pParams->GetGamma());

// Create and load the key look up table.
char key_lut = new char[256];
::load_key_lut(key_lut, m_pParams->GetGain());

// Create and load the detail look up table.
float detail_lut = new float[256];
::load_detail_lut(detail_lut, m_pParams->GetLutScale());

// Determine which bit array to use for the reader's "crude metric"
// computation. If we have just signed this image, then use the
// with message bit array. Otherwise, we are trying to read
// without knowing the true message, and use the estimated
// message for computation of the metric.
unsigned char *referenceBitArray;
if (m_state == IMAGE_SIGNED || m_state == IMAGE_SIGNED_AND_VERIFIED ||
    m_state == IMAGE_SIGNED_AND_SAVED)
    referenceBitArray = m_pPackedMsg->getMsgBitArray();
else
    referenceBitArray = m_pPackedMsg->getReaderBitArray();

long data_length = signedImage.GetXDim() * signedImage.GetYDim();
long x_offset = 0;
long y_offset = 0;
int x_dim = signedImage.GetXDim();
int y_dim = signedImage.GetYDim();

if (signedImage.GetBitsPerPixel() == 8)
    num_channels = 1;
else if (signedImage.GetBitsPerPixel() == 24)
    num_channels = 3;

// See if we should use the super reader.
if (use_super_reader)
    else reading_mode = 1;

else reading_mode = 0;

// Call the core recognizer
::read_8bit_single_channel_or_color(
    signedImage.GetPackedData(),
    x_dim,
    y_dim,
    x_offset,
    y_offset,
    x_dim, // segment is full image.
    y_dim,
    m_pPackedMsg->getMsgBitArrayLength(),
    snowImage.GetPackedData(),
    data_length,
    key_lut,
    luminance_lut,
    detail_lut,
    NULL, // No thumbnail at this time
    //unsignedImage.GetPackedData(),
    NULL, // Don't pass original data now
    (const unsigned char *) referenceBitArray,
    &m_crude_metric,
    &m_range,
    m_pPackedMsg->getReaderBitArray(),
    num_channels,
    reading_mode,
    m_pParams->GetBumpSize());

// Convert the recovered message bits back to an ASCII string.
m_pPackedMsg->BitsToString();

TRACE ("The recognizer detected the following string: %s\n",
    m_pPackedMsg->getRecoveredAsciiMsg());

delete () luminance_lut;
delete () key_lut;
delete () detail_lut;
}

// CDbDoc commands

```

```

// OnSettingsSigner()
// This function is invoked when the user selects the Settings-->
// Signer Controls... menu item. It creates a Signer parameters
// dialog object and presents it to the user as a modal dialog.
// If the user presses OK, we then gather the new parameter values
// and use them to sign the image. Finally, a new view and window
// are created to display the signed image, if no such view already
// exists.
// CDbDoc::OnSettingsSigner()
{
    ParamsDlg dlg;
    CRect rect;
    unsigned old_key;
    BOOL new_user_key = FALSE;

    // Check to see if we are in a legal state for signing.
    if (m_state == NO_IMAGE)
    {
        MessageBox(NULL,
            "An 8 or 24 bit image must be loaded before using the Signer.",
            "Digitarc Signer Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // int scroll_pos

    // Initialize the dialog data
    dlg.m_message = m_pParams->GetMessage();
    dlg.m_gain_from_edit_box = m_pParams->GetGain();
    // dlg.m_gamma = m_pParams->GetGamma(); gamma no longer user cntrl
    dlg.m_key = m_pParams->GetKey();
    old_key = m_pParams->GetKey();
    dlg.m_bump_size = m_pParams->GetBumpSize();
    dlg.m_detail_lut_scale = m_pParams->GetLutScale();

    // Get the coordinates for the scroll bar object window.
    // dlg.m_gain.GetWindowRect(&rect);

    // Try to "create" the scroll bar.
    // dlg.m_gain.Create(WS_CHILD, CRect(10, 50, 200, 20), &dlg, IDC_GAIN);

    // Invoke the dialog box
    if (dlg.DoModal() == IDOK)
    {
        // retrieve the dialog data
        m_pParams->SetMessage(dlg.m_message);
        if (dlg.m_key != old_key)
        {
            m_pParams->SetKey(dlg.m_key);
            new_user_key = TRUE;
        }
        m_pParams->SetGain(dlg.m_gain_from_edit_box);
        m_pParams->SetBumpSize(dlg.m_bump_size);
        m_pParams->SetLutScale(dlg.m_detail_lut_scale);
        // m_pParams->SetGamma(dlg.m_gamma); gamma no longer user cntrl
        // scroll_pos = dlg.m_gain.GetScrollPos();
        // TRACE("Scrollbar position: %d\n", scroll_pos);
        // This is going to take awhile
        BeginWaitCursor();

        // NOTE: AT THIS POINT SHOULD DETERMINE WHAT IMAGE IS IN THE
        // ACTIVE VIEW, AND IF IT CONTAINS A BITMAP SIGN THAT IMAGE.
        // SEE OnSettingsReader(), which uses the correct logic.
        // Then, call MakeSnow(hImageToSignDB) and Sign(hImageToSignDB)
        // If the user seed has changed, or if we haven't yet created
        // a coextensive key, create a snow image.
        if (new_user_key || m_snowDB == NULL)
            MakeSnow(m_hOriginalDB);

        // Use the new settings, and sign the image.
        Sign();

        m_state = IMAGE_SIGNED;

        if (((CDiblookApp *)afxGetApp())->m_autoread)

```

```

(
    // Run the reader again to see if we recover message.
    Read(m_SignedDIB, FALSE);
}
m_state = IMAGE_SIGNED_AND_VERIFIED;

// Now see if a "signed image" view exists. If not, create it.
CreateUniqueView(SIGNED_VIEW);

// Now see if a "status image" view exists. If not, create it.
CdbView *p_statusview;
p_statusview = (CdbView *) CreateUniqueView(STATUS_VIEW);
EndWaitCursor();

// Refresh all of the views (Don't actually need to refresh Original one)
p_statusview->DoResize();
UpdateAllViews(NULL);

// Some debug stuff related to checksums.
TRACE("Signer checksum: %x\n", (int) m_PackedMsg->GetSignerChecksum());
TRACE("Read checksum: %x\n", (int) m_PackedMsg->GetReaderChecksum());
TRACE("Reader computed checksum: %x\n",
      (int) m_PackedMsg->GetComputedReaderChecksum());
)

// CreateUniqueView()
// This function creates a new view of the indicated type, if and
// only if one does not already exist. It returns a pointer to
// the new view, if a new one is created, or a pointer to the
// pre-existing view of the specified type if one already exists.
// The "view type" argument is one of the view types from SignView.h,
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW.
// CView* CDbDoc::CreateUniqueView(int view_type)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView(pos);
        // If we find it, we return the pointer and we're done.
        if ((CdbView*)pView->GetViewType() == view_type)
            return pView;
    }

    // The desired type of view doesn't exist, so we create it.
    CMainFrame *mainFrame = (CMainFrame *) AfxGetApp()->m_pMainWnd;
    mainFrame->MyOnWindowNew();

    // Now find the newly created view (last in list) and set its type.
    pos = GetFirstViewPosition();
    while (pos != NULL)
    {
        pView = GetNextView(pos);
        ((CdbView*)pView)->SetViewType(view_type);
        return(pView);
    }

// ChangeViewType()
// This function finds the view of the "old type", and changes its
// type to "new type". If successful, it returns a pointer to
// the newly changed view. If not, returns NULL.
// The "view type" arguments are from the view types in SignView.h,
// i.e. SIGNED_VIEW, ORIGINAL_VIEW, STATUS_VIEW, ALIGNED_VIEW.
// CView* CDbDoc::ChangeViewType(int old_type, int new_type)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView* pView;
    while (pos != NULL)
    {
        pView = GetNextView(pos);
        // If we find it, change its type we return the pointer and we're done.
        if ((CdbView*)pView->GetViewType() == old_type)
        {
            (CdbView*)pView->SetViewType(new_type);
            return pView;
        }
    }
}
(CdbView*)pView->SetViewType(new_type);
return pView;
}

// We get here only if we failed to find a view of "old_type"
return NULL;
}

// OnSettingsAutoPrint()
// When the user toggles the "Auto-print Report" item in
// the Options menu, this function is invoked. It simply
// toggles the corresponding member variable.
void CDbDoc::OnSettingsAutoPrint()
{
    if (m_autoprint == TRUE)
        m_autoprint = FALSE;
    else
        m_autoprint = TRUE;
}

// OnUpdateSettingsAutoPrint()
// The framework calls this function whenever it is about
// to display the pull-down menu containing the AutoPrint
// Report option. Based on our internal state variable
// m_autoprint, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
void CDbDoc::OnUpdateSettingsAutoPrint(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (m_autoprint == TRUE)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

// OnSettingsReader()
// Invoked when the user selects the Controls->Reader...
// menu option. Presents a ReadParmsDlg dialog object, and
// deals with the operators inputs. On OK, the Read() function
// is called to use the current parameters and run the recog-
// nition core algorithms to try to detect an embedded
// digimarc message.
void CDbDoc::OnSettingsReader()
{
    ReadDlg dlg;
    rect;
    unsigned old_key;
    BOOL new_user_key = FALSE;
    int view_type;
    HDIB hImageToReadDIB;

    // Check to see if we are in a legal state for reading.
    if (m_state == NO_IMAGE)
    {
        MessageBox(NULL,
            "An 8 or 24 bit image must be loaded before using the Reader.",
            "Digimarc Signer Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Determine the type of the active window
    view_type = GetActiveViewType();

    // If active window is not acceptable for reading, warn user & return
    if (view_type != ORIGINAL_VIEW &&
        view_type != SIGNED_VIEW &&
        view_type != ALIGNED_VIEW)
    {
        MessageBox(NULL,
            "The active window must contain an image to be read.",
            "Warning",
            MB_ICONINFORMATION | MB_OK);
        return;
    }

    // Set pointer to the image which is to be read.
    if (view_type == ORIGINAL_VIEW)

```



```

hImageToReadDIB = m_hOriginalDIB;
else if (view_type == SIGNED_VIEW)
hImageToReadDIB = m_hSignedDIB;
else if (view_type == ALIGNED_VIEW)
hImageToReadDIB = m_pAlignedImage->GetHDIIB();
else
{
    MessageBox(NULL, "Bug in OnSettingsReader!", "Error", MB_OK);
    return;
}

// Initialize the dialog data
dlg.m_user_key = m_pParams->GetKey();
old_key = m_pParams->GetKey();
dlg.m_msg_length = m_pParams->GetLength();
dlg.m_gain = m_pParams->GetGain();
dlg.m_bump_size = m_pParams->GetBumpSize();
dlg.m_detail_lut_scale = m_pParams->GetLutScale();
// dlg.m_use_super_reader = m_pParams->GetSuperReaderFlag();

// Invoke the dialog box
if (dlg.DoModal() == IDOK)
{
    m_pParams->SetGain(dlg.m_gain);
    m_pParams->SetBumpSize(dlg.m_bump_size);
    m_pParams->SetLutScale(dlg.m_detail_lut_scale);
    // m_pParams->SetSuperReaderFlag(dlg.m_use_super_reader);

    // If signer has not yet been used, or length changes, need a msg.
    if (m_pParams->GetMessage().GetLength() != (int) dlg.m_msg_length)
    {
        // Create a dummy msg of all x's.
        CString dummy_msg = CString('x', dlg.m_msg_length);
        m_pParams->SetMessage(dummy_msg);
    }

    // Create a PackedMsg object w/ our dummy msg.
    if (m_pPackedMsg != NULL)
        delete m_pPackedMsg;
    m_pPackedMsg = new PackedMsg( (const char *) m_pParams->GetMessage());

    if (dlg.m_user_key != old_key)
    {
        m_pParams->SetKey(dlg.m_user_key);
        new_user_key = TRUE;
    }

    // This is going to take awhile
    BeginWaitCursor();

    // If the user seed has changed, or if we haven't yet created
    // a coextensive key, create a snow image.
    if (new_user_key || m_hSnowDIB == NULL)
        MakeSnow(hImageToReadDIB);

    // Run the reader and attempt to recover message, and compute metrics.
    Read(hImageToReadDIB, m_pParams->GetSuperReaderFlag());

    // Make the state transition: depends on which image was read.
    if (view_type == ORIGINAL_VIEW || view_type == ALIGNED_VIEW)
        m_state = SUSPECT_READ;
    else if (view_type == SIGNED_VIEW)
    {
        if (m_state != IMAGE_SIGNED_AND_SAVED)
            m_state = IMAGE_SIGNED_AND_VERIFIED;
    }

    // KLUDGES for debug. Need the signer timestamp set.
    WHY? 11/24
    m_pParams->UpdateSignatureTime();

    // Now see if a "status image" view exists. If not, create it.
    CDbView *p_statusView;
    p_statusView = (CDbView *) CreateUniqueView(STATUS_VIEW);
    EndWaitCursor();

    // Refresh all of the views (Don't actually need to refresh Original one)
    p_statusView->DoResize();
    UpdateAllViews(NULL);

    // See if the checksum read and the checksum computed from the
    // read message string agree. If not, warn user.
    if (m_pPackedMsg->GetReaderChecksum() !=
        m_pPackedMsg->GetComputedReaderChecksum())
    {
        MessageBox(NULL,
            "The embedded checksum didn't match the computed checksum.",
            "Warning", MB_OK);
    }
}
}

}

// Find the active view, determine its type, and return
// it to the caller. The type is one of those listed
// in the DIBView.h file.
int CDbDoc::GetActiveViewType(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView *pview;
    while (pos != NULL)
    {
        pview = GetNextView(pos);

        // If we find it, we return the pointer and we're done.
        if ( (CDbView*)pview->IsActive() == TRUE)
            return ((CDbView*)pview)->GetViewType();
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDbDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(UNKNOWN_VIEW);
}

// Return a pointer to the active view (i.e., a CDbView*, or NULL
// if something goes wrong.
CDbView * CDbDoc::GetActiveView(void)
{
    BOOL view_found = FALSE;
    POSITION pos = GetFirstViewPosition();
    CView *pview;
    while (pos != NULL)
    {
        pview = GetNextView(pos);

        // If we find it, we return the pointer and we're done.
        if ( (CDbView*)pview->IsActive() == TRUE)
            return (CDbView*)pview;
    }

    // We can get here when other apps are running and Windows sends message
    // resulting in CDbDoc::OnUpdateFileSaveAs() being called.
    // MessageBox(NULL, "Error in GetActiveViewType!", "Error", MB_OK);
    return(NULL);
}

// OnSettingsAutoread()
// When the user toggles the "Auto-read after Signing" item in
// the Options menu, this function is invoked. It simply
// toggles the corresponding member variable.
// We currently also toggle the application level variable,
// so that the settings are global to all docs.
void CDbDoc::OnSettingsAutoread()
{
    if (m_autoread == TRUE)
    {
        m_autoread = FALSE;
        ((CDbLookAndFeel *)AfxGetApp()->m_autoread = FALSE;
    }
    else
    {
        m_autoread = TRUE;
        ((CDbLookAndFeel *)AfxGetApp()->m_autoread = TRUE;
    }
}

// OnUpdateSettingsAutoread()
// The framework calls this function whenever it is about
// to display the pulldown menu containing the Autoread
// option. Based on our internal state variable

```

```

// m_autoread, we set or clear the check mark next to
// the menu item using the pCmdUI->SetCheck() function.
// OnSetingsAlign()
void CDbDoc::OnUpdateSettingsAutoread(CCmdUI* pCmdUI)
{
    // Set or clear the check mark in the menu
    if (((CDbLookApp *)AfxGetApp())->m_autoread == TRUE)
        pCmdUI->SetCheck(TRUE);
    else
        pCmdUI->SetCheck(FALSE);
}

// OnSetingsAlign()
// This function is called when the user selects the "Align" menu option.
// A CFileDialog object is created and used in order for the operator
// to specify the name of the "Reference Image" (a signed or unsigned
// original image used as the template).
// void CDbDoc::OnSetingsAlign()
{
    CString refname;
    BOOL success_flag;

    // Create a filter for the types of files the file dialog will offer
    char szFilter[] =
        "Windows Bit Map Files (*.bmp)|*.bmp|Device Independent Bitmaps (*.dib)|*.dib|*.*|All Files (*.*)|*.*||";

    // Construct a file dialog
    CFileDialog fileDlg(TRUE, // its a file open (not save) dialog
        NULL, // B.M.P.
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
        szFilter);

    // Over-ride the default title in the file dialog window
    fileDlg.m_ofn.lpstrTitle = "Select a template file to be used for alignment";

    // Display the file dialog
    if (fileDlg.DoModal() == IDOK)
    {
        // Get the name of the reference image file.
        refname = fileDlg.GetPathName();

        BeginWaitCursor();

        // Create an Image object for the reference image.
        // If one already exists, delete it first.
        if (m_pRefImage != NULL)
            delete m_pRefImage;

        m_pRefImage = new Image(refname);

        if (m_pRefImage->GetFileOK == FALSE) // bail out if something went wrong
            return;

        // Display the reference image
        CreateUniqueView(REF_VIEW);

        // UpdateAllViews(NULL);

        TRACE("Call the Align() function (this is a test of trace output.)\n");

        // Do the actual alignment and change update the state description.
        success_flag = Align_it();

        if (success_flag)
        {
            m_state = SUSPECT_ALIGNED;

            // Now, the template image object has had its packed data array replaced
            // by the aligned co-extensive image. Need to move this packed data
            // into the DIB array for display (and possible file saving) purposes.
            m_pRefImage->UnpackData();

            // We now call the image the Aligned image, not reference
            m_pAlignedImage = m_pRefImage;
            m_pRefImage = NULL;

            CreateUniqueView(ALIGNED_VIEW);

            // Create a status view, if it doesn't already exist.
            CDbView* p_statusView;
            p_statusView = (CDbView *) CreateUniqueView(STATUS_VIEW);

            p_statusView->DoResize();

            UpdateAllViews(NULL);
        }
    }
}

// EndWaitCursor();
}

// This function is responsible for carrying out the alignment operation,
// by calling upon Geoff's core algorithms. It is assumed that on entry
// 1) m_horiginaidib is DIB of the suspect image, already loaded.
// 2) m_pRefImage points to a Image object with the template (or
// reference) image.
// void CDbDoc::Align_it(void)
{
    int num_channels;

    // Create an image object for the suspect image.
    Image suspectImage(m_horiginaidib);

    // Currently we require that the reference and suspect are of same type
    // (i.e., both color or B&W).
    if (suspectImage.GetBitsPerPixel() != m_pRefImage->GetBitsPerPixel())
    {
        MessageBox(NULL,
            "The suspect and reference images must both be color or B&W",
            "Warning",
            MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }

    // Construct Align object.
    if (m_pAlign != NULL)
        delete m_pAlign;

    m_pAlign = new Align;

    // Create the "byte-wise" packed data arrays from the DIB 4-byte packing
    suspectImage.MakePackedData();
    m_pRefImage->MakePackedData();

    if (suspectImage.GetBitsPerPixel() == 8)
        num_channels = 1; // B&W image
    else if (suspectImage.GetBitsPerPixel() == 24)
        num_channels = 3; // Color image

    // Call the core algorithm to do the alignment.
    m_pAlign->direct_registration(m_pRefImage->GetPackedData(),
        m_pRefImage->GetXDIm(),
        m_pRefImage->GetYDIm(),
        suspectImage.GetPackedData(),
        suspectImage.GetXDIm(),
        suspectImage.GetYDIm(),
        num_channels);

    return(TRUE);
}

// OnUpdateFilesSaveAs()
// When the File pulldown menu is selected, this function is called
// upon to determine whether the "Save As..." menu item should be
// enabled. It determines the type of the current view, and if it
// is of a type for which we currently allow file saves, the menu
// item is enabled.
// void CDbDoc::OnUpdateFilesSaveAs(CCmdUI* pCmdUI)
{
    int view_type;

    // Determine the type of the current view.
    view_type = GetActiveViewType();

    // If the active view contains an image, we know how to save it.
    if (view_type == ORIGINAL_VIEW ||
        view_type == SIGNED_VIEW ||
        view_type == ALIGNED_VIEW ||
        view_type == STATUS_VIEW)
    {
        pCmdUI->Enable(TRUE);
    }
    else
        pCmdUI->Enable(FALSE);
}

```

```

    pCmdUI->Enable(FALSE);
}

// Implementation
protected:
    virtual ~CDibDoc();

    virtual BOOL OnSaveDocument(const char* pszPathName);
    virtual BOOL OnOpenDocument(const char* pszPathName);

    //void OnEditSettings();

private:
    void MangleDIB(void);
    void CDibDoc::DumpBitmapInfoHeader() const;
    void MakeSnow(HDIB hParentDIB);
    void SignDoc();
    void Read(HDIB hSignedDIB, BOOL use_super_reader);
    BOOL Align_It(void);
    CView* CreateOnQueueView(int view_type);
    CView* ChangeViewType(int old_type, int new_type);
    int GetActiveViewType(void);
    CDibView* GetActiveView(void);

    int m_state;
    CString m_filename;

    float m_crude_metric;
    float m_range;

    Image* m_pRefImage;
    Image* m_pAlignedImage;

    Align* m_pAlign;

protected:
    // Obsolete
    // HDIB m_hDIB;
    // CPalette* m_palDIB;
    // CSize m_sizeDoc;
    // int m_BitsPerPixel;

    CView* m_pSignedView;

    // Ptr to the initially loaded image, unmodified by signing.
    HDIB m_hOriginalDIB;

    // Add additional DIB handles for the snow image and signed image.
    HDIB m_hSnowyDIB;
    HDIB m_hSignedDIB;

    // Need to know total space needed for these guys.
    DWORD m_dwTotalDIBSize;

    // Pointer to parameters object.
    SignerParams* m_pParams;

    PackedMsg* m_pPackedMsg;

    BOOL m_autoprint;
    BOOL m_autoread;

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    virtual BOOL OnNewDocument();

    // Generated message map functions
protected:
    ///({AFX_MSG(CDibDoc)
    afx_msg void OnSettingsSigner();
    afx_msg void OnSettingsAutoprint();
    afx_msg void OnUpdateSettingsAutoprint(CCmdUI* pCmdUI);
    afx_msg void OnSettingsReader();
    afx_msg void OnSettingsAutoread();
    afx_msg void OnUpdateSettingsAutoread(CCmdUI* pCmdUI);
    afx_msg void OnSettingsAlign();
    afx_msg void OnUpdateFilesaveAs(CCmdUI* pCmdUI);
    ///})AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////

```

SIGNDOC.H

```

// DESCRIPTION:
// Interface file for the CDibDoc class. This defines the document class
// for the Signer. Under the Microsoft Foundation Class (MFC) architecture,
// the Document/View model is the preferred method. This header file
// defines our additions to the generic Document class created by the
// Visual C++ wizards.

// Copyright (C) 1996 Digimarc Corporation, all rights reserved.

////////////////////
#include "dibapi.h"
#include "packmsg.h"
#include "params.h"
#include "image.h"
#include "Align.h"

#include "signview.h"

// Define the possible states...
#define NO_IMAGE 0
#define IMAGE_LOADED 1
#define IMAGE_SIGNED 2
#define IMAGE_SIGNED_AND_VERIFIED 3
#define SUSPECT_READ 4
#define IMAGE_SIGNED_AND_SAVED 5
#define SUSPECT_ALIGNED 6

#define FORCE_TO_1_CHANNEL TRUE // For clarity when packing rgb images to 1 chan.

class CDibView;

class CDibDoc : public CDocument
{
protected: // create from serialization only
    CDibDoc();
    DECLARE_DYNCREATE(CDibDoc)

    // Attributes
public:
    // HDIB GetHDIB() const
    // { return m_hDIB; }

    HDIB GetSignedHDIB() const
    { return m_hSignedDIB; }
    HDIB GetOriginalHDIB() const
    { return m_hOriginalDIB; }
    HDIB GetSnowyHDIB() const
    { return m_hSnowyDIB; }
    HDIB GetRefHDIB() const
    { return m_pRefImage->GetHDIB(); }
    HDIB GetAlignedHDIB() const
    { return m_pAlignedImage->GetHDIB(); }

    CPalette* GetDocPalette() const
    { return m_palDIB; }
    CSize GetDocSize() const
    { return m_sizeDoc; }

    PackedMsg* GetPackedMsg() const
    { return m_pPackedMsg; }

    SignerParams* GetSignerParams() const
    { return m_pParams; }

    int GetState() const { return m_state; }

    const CString& GetFilename() const { return m_filename; }

    float GetMetric() const { return m_crude_metric; }
    float GetRange() const { return m_range; }

    // Accessors so view objects can get alignment results.
    const AlignStatus GetAlignStatus(void) const { return m_pAlign->GetAlignStatus(); }

    // Operations
public:
    void ReplaceHDIB(HDIB hDIB);
}

```

```

// signer.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "signer.h"

#include "mainfrm.h"
#include "signdoc.h"
#include "signview.h"
#include "mychildw.h"

// #include "AFXPRIV.H"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

char *global_cmd_line_args;
///////////////////////////////////////////////////////////////////
// CDibLookApp
//
BEGIN_MESSAGE_MAP(CDibLookApp, CWinApp)
//({AFX_MSG_MAP(CDibLookApp)
// ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
//})AFX_MSG_MAP
// Standard file based document commands
// ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
// ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
// ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////
// CDibLookApp construction
// Place all significant initialization in InitInstance
//
CDibLookApp::CDibLookApp()
{
    m_lpParams = NULL;
    m_autoread = FALSE;
}

CDibLookApp::~CDibLookApp()
{
    if (m_lpParams != NULL)
        delete m_lpParams;
}

// The one and only CDibLookApp object
//
CDibLookApp NEAR theApp;

// CDibLookApp initialization
//
BOOL CDibLookApp::InitInstance()
{
    // Standard initialization
    // (if you are not using these features and wish to reduce the size
    // of your final executable, you should remove the following initialization
    // SetDialogBkColor(); // set dialog background color
    LoadStdProfileSettings(); // Load standard INI file options (including MRU)

    // Register document templates which serve as connection between
    // documents and views. Views are contained in the specified view
    AddDocTemplate(new CMultiDocTemplate(IDR_DIBTYPEB,
        RUNTIME_CLASS(CDibDoc),
        RUNTIME_CLASS(CMyChildWnd), // I replace CMyChildWnd
        RUNTIME_CLASS(CDibView)));

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    m_pMainWnd = pMainFrame;

    // enable file manager drag/drop and DDE Execute open
    m_pMainWnd->DragAcceptFiles();
    EnableShellOpen();
    RegisterShellFileTypes();

    ///////////////////////////////////////////////////////////////////
    // SIGNER.H
    // signer.h : main header file for the SIGNER application
    //
    // As a test, save a global copy of command line args
    // global_cmd_line_args = m_lpCmdLine;
    m_lpParams = new SignerParams(m_lpCmdLine);

    // DEBUG: display the command line before we parse it.
    // AfxMessageBox(m_lpCmdLine);

    // simple command line parsing
    if (m_lpParams->GetInputFilename() == NULL)
    {
        // create a new (empty) document
        // OnFileNew();
    }
    else if ((m_lpCmdLine[0] == '-' || m_lpCmdLine[0] == '/') &&
        (m_lpCmdLine[1] == 'e' || m_lpCmdLine[1] == 'g'))
    {
        // program launched embedded - wait for DDG or OLE open
    }
    else
    {
        // open an existing document
        OpenDocumentFile(m_lpParams->GetInputFilename());
    }

    // TRY adding another window.
    // pMainFrame->OnWindowNew();
    // pMainFrame->SendMessage(ID_WINDOW_NEW);
    // pMainFrame->OnWindowNewTest();
    return TRUE;
}

// CAboutDlg dialog used for App About
//
class CAboutDlg : public CDialog
{
public:
    CAboutDlg() : CDialog(CAboutDlg::IDD)
    {
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
    }

    // Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
// No message handlers
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CDibLookApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// CDibLookApp commands
//
///////////////////////////////////////////////////////////////////

```

```

#ifdef _AFXWIN_H
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h" // main symbols
#include "params.h"

#define WM_DOREALIZE (WM_USER + 0)

// CDialogApp
// See dialog.cpp for the implementation of this class
class CDialogApp : public CWinApp
{
public:
    CDialogApp();
    ~CDialogApp();

    // Create a command line parameter object.
    SignerParams m_lpParams;
    SignerParams *GetParams(void) {return m_lpParams;}

    BOOL m_autoread;

// Overrides
    virtual BOOL InitInstance();
// Implementation
    //((AFX_MSG(CDialogApp)
    afx_msg void OnAppAbout();
    //))AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// Microsoft Developer Studio generated resource script.
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
// Generated from the TEXTINCLUDE 2 resource.
#include "afxres.h"

#ifdef _AFXDLL
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif // _AFXDLL

// Defining APSTUDIO_INVOKED
// TEXTINCLUDE
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include \"afxres.rc\"\\r\\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include \"afxres.rc\"\\r\\n"
    "#include \"afxprint.rc\"\\r\\n"

```

```

    "\0"
END

// APSTUDIO_INVOKED
// Icon
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME ICON DISCARDABLE "RES\\DIBLOCK.ICO"
IDR_DIBTYPE ICON DISCARDABLE "RES\\DIBDOC.ICO"
// Bitmap
// IDR_MAINFRAME BITMAP MOVABLE PURE "RES\\TOOLBAR.BMP"
// Menu
// IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "File"
    BEGIN
        MENUITEM "<New\\tCtrl+N",
        ID_FILE_NEW
        MENUITEM "<Open...\\tCtrl+O",
        ID_FILE_OPEN
        MENUITEM "<Print Setup...",
        ID_FILE_PRINT_SETUP
        MENUITEM "<Recent File",
        ID_FILE_MRU_FILE1, GRAYED
        MENUITEM "<Separator",
        ID_APP_EXIT
    END
    POPUP "<View"
    BEGIN
        MENUITEM "<Toolbar",
        ID_VIEW_TOOLBAR
        MENUITEM "<Status Bar",
        ID_VIEW_STATUS_BAR
    END
    POPUP "<Help"
    BEGIN
        MENUITEM "<About SIGNER...",
        ID_APP_ABOUT
    END
END

IDR_DIBTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "File"
    BEGIN
        MENUITEM "<New\\tCtrl+N",
        ID_FILE_NEW
        MENUITEM "<Open...\\tCtrl+O",
        ID_FILE_OPEN
        MENUITEM "<Close",
        ID_FILE_CLOSE
        MENUITEM "<Save <as...",
        ID_FILE_SAVE_AS
        MENUITEM "<Print...\\tCtrl+P",
        ID_FILE_PRINT
        MENUITEM "<Print Preview",
        ID_FILE_PRINT_PREVIEW
        MENUITEM "<Print Setup...",
        ID_FILE_PRINT_SETUP
        MENUITEM "<Separator",
        ID_FILE_MRU_FILE1, GRAYED
        MENUITEM "<Recent File",
        ID_APP_EXIT
        MENUITEM "<Exit",
        ID_APP_EXIT
    END
    POPUP "<Edit"
    BEGIN
        MENUITEM "<Undo\\tCtrl+Z",
        ID_EDIT_UNDO
        MENUITEM "<Separator",
        ID_EDIT_SEPARATOR
        MENUITEM "<Cut\\tCtrl+X",
        ID_EDIT_COPY
        MENUITEM "<Copy\\tCtrl+C",
        ID_EDIT_PASTE
        MENUITEM "<Paste\\tCtrl+V",
        ID_EDIT_PASTE
    END
    POPUP "<Actions"
    BEGIN
        MENUITEM "<Sign...",
        ID_SETTINGS_SIGNER
        MENUITEM "<Align...",
        ID_SETTINGS_ALIGN
        MENUITEM "<Read...",
        ID_SETTINGS_READER
    END
    POPUP "<Window"
    BEGIN
        MENUITEM "<New Window",
        ID_WINDOW_NEW
        MENUITEM "<Cascade",
        ID_WINDOW_CASCADE
        MENUITEM "<Tile",
        ID_WINDOW_TILE_HORZ
        MENUITEM "<Arrange Icons",
        ID_WINDOW_ARRANGE
    END
END

```

```

POPUP "zView"
BEGIN
    MENUITEM "zToolbar",
    MENUITEM "zStatus Bar",
    MENUITEM SEPARATOR,
    MENUITEM "Signed Image",
    MENUITEM "Unsigned Image",
    MENUITEM "Code Pattern",
    MENUITEM "Status",
END
POPUP "zOptions"
BEGIN
    MENUITEM "Auto-read After Signing",
    MENUITEM "Registry...",
    MENUITEM "Auto-print Report",
END
POPUP "zHelp"
BEGIN
    MENUITEM "zAbout SIGNER...",
END
END

ID_VIEW_TOOLBAR
ID_VIEW_STATUS_BAR
ID_VIEW_SIGNED
ID_VIEW_UNSIGNED
ID_VIEW_SHOW_IMAGE
ID_VIEW_STATUS

ID SETTINGS_AUTOREAD
ID SETTINGS_REGISTRY, GRAYED
ID SETTINGS_AUTOPRINT

ID_APP_ABOUT

// String Table
//
//
STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME "Digimarc Signer Application"
    IDR_DIBTYPE "(\nSIGNER Document\NBMP Files"
    END
    (*.bmp) (\n.bmp\nSignerFileType\nSIGNER File Type"
    END
STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE "Digimarc Signer Application"
    AFX_IDS_IDLEMESSAGE "Ready"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EYT "EXT"
    ID_INDICATOR_CAPS "CAP"
    ID_INDICATOR_NUM "NUM"
    ID_INDICATOR_SCRL "SCRL"
    ID_INDICATOR_OVR "OVR"
    ID_INDICATOR_REC "REC"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW "Create a new document"
    ID_FILE_OPEN "Open an existing document"
    ID_FILE_CLOSE "Close the active document"
    ID_FILE_SAVE "Save the active document"
    ID_FILE_SAVE_AS "Save the signed image with a new name"
    ID_FILE_PAGE_SETUP "Change the printing options"
    ID_FILE_PRINT_SETUP "Change the printer and printing options"
    ID_FILE_PRINT "Print the active document"
    ID_FILE_PRINT_PREVIEW "Display full pages"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT "Display program information, version number and copyright"
    ID_APP_EXIT "Quit the application; prompts to save documents"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILTER1 "Open this document"
    ID_FILE_MRU_FILTER2 "Open this document"
    ID_FILE_MRU_FILTER3 "Open this document"
    ID_FILE_MRU_FILTER4 "Open this document"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE "Switch to the next window pane"
    ID_PREV_PANE "Switch back to the previous window pane"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW "Open another window for the active document"
    ID_WINDOW_ARRANGE "Arrange icons at the bottom of the window"
    ID_WINDOW_CASCADE "Arrange windows so they overlap"
    ID_WINDOW_TILE_HORZ "Arrange windows as non-overlapping tiles"
    ID_WINDOW_TILE_VERT "Arrange windows as non-overlapping tiles"
    ID_WINDOW_SPLIT "Split the active window into panes"
    END
STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR "Erase the selection"
    END

```

```

ID_EDIT_CLEAR_ALL
ID_EDIT_COPY
ID_EDIT_CUT
ID_EDIT_FIND
ID_EDIT_PASTE
ID_EDIT_REPEAT
ID_EDIT_REPLACE
ID_EDIT_SELECT_ALL
ID_EDIT_UNDO
ID_EDIT_REDO

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR
    ID_VIEW_STATUS_BAR
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE
    AFX_IDS_SCMOVE
    AFX_IDS_SCHINIMIZE
    AFX_IDS_SCHMAXIMIZE
    AFX_IDS_SCHEXTWINDOW
    AFX_IDS_SCPREVIEWWINDOW
    AFX_IDS_SCCLOSE
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE
    AFX_IDS_SCTASKLIST
    AFX_IDS_MDCHILD
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_SETTINGS
    ID_VIEW_SIGNED
    ID_VIEW_UNSIGNED
    ID_VIEW_SNOW
    ID_VIEW_SNOWY_IMAGE
    ID_VIEW_STATUS
    ID_SETTINGS_SIGNER
    ID_SETTINGS_READER
    ID_SETTINGS_REGISTRY
    ID_SETTINGS_AUTOPRINTREPORT
    ID_SETTINGS_AUTOPRINT
    ID_OPTIONS_AUTOREAD
    ID_SETTINGS_AUTOREAD
    ID_CONTROLS_ALIGN
    ID_SETTINGS_ALIGN
END

#endif // English (U.S.) resources
// ~~~~~

#ifndef APSTUDIO_INVOKED
// ~~~~~
// Generated from the TEXTINCLUDE 3 resource.
#include "afxres.rc"
#include "afxprint.rc"
// ~~~~~
#endif // not APSTUDIO_INVOKED

SIGNERW1.MAK

# Microsoft Developer Studio Generated NMAKE File, Format Version 4.00
# ** DO NOT EDIT **

# TARGETTYPE "Win32 (x86) Application" 0x0101

CFG="$(CFG)" == ""
CFG=Signer - Win32 Debug
MESSAGE No configuration specified. Defaulting to Signer - Win32 Debug.
ENDIF

IF "$(CFG)" == "Signer - Win32 Release" && "$(CFG)" != "Signer - Win32 Debug"
MESSAGE Invalid configuration "$(CFG)" specified.
MESSAGE You can specify a configuration when running NMAKE on this makefile

```

```

MESSAGE by defining the macro CFG on the command line. For example:
MESSAGE NMAKE /f "SignerWin32.mak" CFG="Signer - Win32 Debug"
MESSAGE Possible choices for configuration are:
MESSAGE "Signer - Win32 Release" (based on "Win32 (x86) Application")
MESSAGE "Signer - Win32 Debug" (based on "Win32 (x86) Application")
ERROR An invalid configuration is specified.
ENDIF

IF "$(OS)" == "Windows_NT"
NULL
ELSE
NULL
ENDIF
#####
# PROP Target_Last_Scanned "Signer - Win32 Debug"
#####
MTLibCtyLib.exe
RSC-rt.exe
CPP-cl.exe

IF "$(CFG)" == "Signer - Win32 Release"
# PROP BASE Use_MFC 1
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 1
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
OUTDIR=. \Release
INTDIR=. \Release

ALL : "$(OUTDIR)\SignerWin32.exe" "$(OUTDIR)\SignerWin32.bsc"

CLEAN :
-erase . \Release\SignerWin32.bsc
-erase . \Release\Mainfrm.sbr
-erase . \Release\Sign.sbr
-erase . \Release\SignDoc.sbr
-erase . \Release\Coxkey.sbr
-erase . \Release\ParamsDlg.sbr
-erase . \Release\Fft.sbr
-erase . \Release\Stdafx.sbr
-erase . \Release\MyChildw.sbr
-erase . \Release\Packmsg.sbr
-erase . \Release\Signview.sbr
-erase . \Release\MyFile.sbr
-erase . \Release\Image.sbr
-erase . \Release\Params.sbr
-erase . \Release\Signer.sbr
-erase . \Release\Align.sbr
-erase . \Release\Read.sbr
-erase . \Release\Dibapi.sbr
-erase . \Release\ReadDlg.sbr
-erase . \Release\SignerWin32.exe
-erase . \Release\Params.obj
-erase . \Release\Signer.obj
-erase . \Release\Align.obj
-erase . \Release\Read.obj
-erase . \Release\Dibapi.obj
-erase . \Release\ReadDlg.obj
-erase . \Release\Mainfrm.obj
-erase . \Release\SignDoc.obj
-erase . \Release\Coxkey.obj
-erase . \Release\ParamsDlg.obj
-erase . \Release\Fft.obj
-erase . \Release\Stdafx.obj
-erase . \Release\MyChildw.obj
-erase . \Release\Packmsg.obj
-erase . \Release\Signview.obj
-erase . \Release\MyFile.obj
-erase . \Release\Image.obj
-erase . \Release\Signer.res

"$(OUTDIR)" :
if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
/c
ADD BASE CPP /nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_MBCS" /FR /YX
/c
ADD CPP /nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_MBCS" /FR /YX /c
CPP PROJ /nologo /MT /W3 /GX /O1 /D "WIN32" /D "NDEBUG" /D "_MBCS" /FR "$(INTDIR)/" /P "$(INTDIR)/SignerWin32.pch" /YX /Fo"$(INTDIR)/" /c

```

```

CPP_OBJS-.\Release\
CPP_SRS-.\Release\
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ-.\nologo /D "NDEBUG" /win32
# ADD BASE RSC /I 0x409 /d "NDEBUG"
RSC_PROJ-.\I 0x409 /d "NDEBUG"
BSC32-bscmake.exe /o "$(OUTDIR)\SignerWin32.res" /d "NDEBUG"
# ADD BASE BSC32
# ADD BSC32 /nologo
BSC32_FLAGS-.\nologo /o "$(OUTDIR)\SignerWin32.bsc"
BSC32_SRS-.\
$(INTDIR)\Mainfrm.sbr \
$(INTDIR)\Sign.sbr \
$(INTDIR)\Signdoc.sbr \
$(INTDIR)\Coxkey.sbr \
$(INTDIR)\Parmadlg.sbr \
$(INTDIR)\Stdafx.sbr \
$(INTDIR)\Mychildw.sbr \
$(INTDIR)\Packmsg.sbr \
$(INTDIR)\Signview.sbr \
$(INTDIR)\Myfile.sbr \
$(INTDIR)\Image.sbr \
$(INTDIR)\Params.sbr \
$(INTDIR)\Signer.sbr \
$(INTDIR)\Align.sbr \
$(INTDIR)\Read.sbr \
$(INTDIR)\Dibapi.sbr \
$(INTDIR)\ReadDlg.sbr \
$(OUTDIR)\SignerWin32.bsc : "$(OUTDIR)" $(BSC32_SRS)
$(BSC32) @<<
$(BSC32_FLAGS) $(BSC32_SRS)
<<
LINK32-link.exe
# ADD BASE LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /machine:IX86
# ADD LINK32 oldnames.lib /nologo /stack:0x4800 /subsystem:windows /machine:IX86
# SUBTRACT LINK32 /profile /debug
LINK32_FLAGS-oldnames.lib /nologo /stack:0x4800 /subsystem:windows /
/inc:$(IntDir)\pobj\ /pdb:"$(OUTDIR)\SignerWin32.pdb" /machine:IX86\
/def:"$(IntDir)\Signer.def" /out:"$(OUTDIR)\SignerWin32.exe"
DEP_FILE-.\
"$(IntDir)\Signer.def"
LINK32_OBJS-.\
$(INTDIR)\Params.obj \
$(INTDIR)\Signer.obj \
$(INTDIR)\Align.obj \
$(INTDIR)\Read.obj \
$(INTDIR)\Dibapi.obj \
$(INTDIR)\ReadDlg.obj \
$(INTDIR)\Mainfrm.obj \
$(INTDIR)\Sign.obj \
$(INTDIR)\Signdoc.obj \
$(INTDIR)\Coxkey.obj \
$(INTDIR)\Parmadlg.obj \
$(INTDIR)\Stdafx.obj \
$(INTDIR)\Mychildw.obj \
$(INTDIR)\Packmsg.obj \
$(INTDIR)\Signview.obj \
$(INTDIR)\Myfile.obj \
$(INTDIR)\Image.obj \
$(INTDIR)\Signer.res"
"$(OUTDIR)\SignerWin32.exe" : "$(OUTDIR)" $(LINK32_OBJS)
$(LINK32) @<<
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ELSEIP "$(CFG)" == "Signer - Win32 Debug"
# PROP BASE Use_MFC 1
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 1
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
OUTDIR-.\Debug
ALL : "$(OUTDIR)\SignerWin32.exe" "$(OUTDIR)\SignerWin32.bsc"
CLEAN :

```

```

-erase " .\Debug\vc40.pdb"
-erase " .\Debug\vc40.idb"
-erase " .\Debug\SignerWin32.bsc"
-erase " .\Debug\Dibapi.sbr"
-erase " .\Debug\ReadDlg.sbr"
-erase " .\Debug\Myfile.sbr"
-erase " .\Debug\Mychildw.sbr"
-erase " .\Debug\Coxkey.sbr"
-erase " .\Debug\Signview.sbr"
-erase " .\Debug\Signer.sbr"
-erase " .\Debug\Stdafx.sbr"
-erase " .\Debug\Packmsg.sbr"
-erase " .\Debug\Read.sbr"
-erase " .\Debug\fft.sbr"
-erase " .\Debug\Sign.sbr"
-erase " .\Debug\Image.sbr"
-erase " .\Debug\Parmadlg.sbr"
-erase " .\Debug\Mainfrm.sbr"
-erase " .\Debug\Signdoc.sbr"
-erase " .\Debug\Align.sbr"
-erase " .\Debug\Params.sbr"
-erase " .\Debug\SignerWin32.exe"
-erase " .\Debug\Params.obj"
-erase " .\Debug\Dibapi.obj"
-erase " .\Debug\ReadDlg.obj"
-erase " .\Debug\Myfile.obj"
-erase " .\Debug\Mychildw.obj"
-erase " .\Debug\Coxkey.obj"
-erase " .\Debug\Signview.obj"
-erase " .\Debug\Signer.obj"
-erase " .\Debug\Stdafx.obj"
-erase " .\Debug\Packmsg.obj"
-erase " .\Debug\Read.obj"
-erase " .\Debug\fft.obj"
-erase " .\Debug\Sign.obj"
-erase " .\Debug\Image.obj"
-erase " .\Debug\Parmadlg.obj"
-erase " .\Debug\Mainfrm.obj"
-erase " .\Debug\Signdoc.obj"
-erase " .\Debug\Align.obj"
-erase " .\Debug\Signer.res"
"$(OUTDIR)" :
if not exist "$(OUTDIR)\$(NULL)" mkdir "$(OUTDIR)"
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D
#_MBCS /FR /YX /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D "_MBCS" /FR
CPP_C
D-PROJ-.\nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
/RS-BCS /FR-$(INTDIR) /Fp-$(INTDIR)\SignerWin32.pch /YX /Fo-$(INTDIR) /
/RC-$(INTDIR) /c
CPP_OBJS-.\Debug\
CPP_SRS-.\Debug\
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ-.\nologo /D "NDEBUG" /win32
# ADD BASE RSC /I 0x409 /d "NDEBUG"
RSC_PROJ-.\I 0x409 /d "NDEBUG"
BSC32-bscmake.exe /o "$(INTDIR)\SignerWin32.res" /d "NDEBUG"
# ADD BASE BSC32 /nologo
BSC32_FLAGS-.\nologo /o "$(OUTDIR)\SignerWin32.bsc"
BSC32_SRS-.\
$(INTDIR)\Dibapi.sbr \
$(INTDIR)\ReadDlg.sbr \
$(INTDIR)\Myfile.sbr \
$(INTDIR)\Mychildw.sbr \
$(INTDIR)\Coxkey.sbr \
$(INTDIR)\Signview.sbr \
$(INTDIR)\Signer.sbr \
$(INTDIR)\Stdafx.sbr \
$(INTDIR)\Read.sbr \
$(INTDIR)\Packmsg.sbr \
$(INTDIR)\fft.sbr \
$(INTDIR)\Sign.sbr \
$(INTDIR)\Image.sbr \
$(INTDIR)\Parmadlg.sbr \
$(INTDIR)\Mainfrm.sbr \
$(INTDIR)\Signdoc.sbr \
$(INTDIR)\Align.sbr \
$(INTDIR)\Params.sbr \
"$(OUTDIR)\SignerWin32.bsc" : "$(OUTDIR)" $(BSC32_SRS)
$(BSC32) @<<
$(BSC32_FLAGS) $(BSC32_SRS)
<<
LINK32-link.exe

```



```

# ADD BASE LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /debug /machine:IX86
# ADD LINK32 oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile /debug /machine:IX86
LINK32_FLAGS=oldnames.lib /nologo /stack:0x2800 /subsystem:windows /profile /debug /machine:IX86
DEP_FILE=.\_Signer.def*
LINK32_OBJS=
$(INTDIR)\Params.obj\
$(INTDIR)\Dibapi.obj\
$(INTDIR)\Readimg.obj\
$(INTDIR)\Myfile.obj\
$(INTDIR)\Mychildw.obj\
$(INTDIR)\Coxkey.obj\
$(INTDIR)\Signview.obj\
$(INTDIR)\Signer.obj\
$(INTDIR)\Stdafx.obj\
$(INTDIR)\Read.obj\
$(INTDIR)\Packmsg.obj\
$(INTDIR)\Pft.obj\
$(INTDIR)\Sign.obj\
$(INTDIR)\Image.obj\
$(INTDIR)\Parmadlg.obj\
$(INTDIR)\Mainfrm.obj\
$(INTDIR)\Signdoc.obj\
$(INTDIR)\Align.obj\
$(INTDIR)\Signer.res*
$(OUTDIR)\SignerWin32.exe : $(OUTDIR) $(DEP_FILE) $(LINK32_OBJS)
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ENDIF
.c{$(CPP_OBJS)}_obj:
$(CPP) $(CPP_PROJ) $<
.cpp{$(CPP_OBJS)}_obj:
$(CPP) $(CPP_PROJ) $<
.cxx{$(CPP_OBJS)}_obj:
$(CPP) $(CPP_PROJ) $<
.c{$(CPP_SBRS)}_sbr:
$(CPP) $(CPP_PROJ) $<
.cpp{$(CPP_SBRS)}_sbr:
$(CPP) $(CPP_PROJ) $<
.cxx{$(CPP_SBRS)}_sbr:
$(CPP) $(CPP_PROJ) $<
#####
# Begin Target
# Name "Signer - Win32 Release"
# Name "Signer - Win32 Debug"
!IF "$(CFG)" == "Signer - Win32 Release"
!ELSEBIP "$(CFG)" == "Signer - Win32 Debug"
!ENDIF
#####
SOURCE=.\Coxkey.cpp
DEP_CPP_COXKB=
"..\Coxkey.h"
"..\Dibapi.h"
"..\Stdafx.h"
"$(INTDIR)\Coxkey.obj" : $(SOURCE) $(DEP_CPP_COXKB) "$(INTDIR)"
"$(INTDIR)\Coxkey.sbr" : $(SOURCE) $(DEP_CPP_COXKB) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Dibapi.cpp
DEP_CPP_DIBAP=
"..\Stdafx.h"
"..\Dibapi.h"
"$(INTDIR)\Dibapi.obj" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"
"$(INTDIR)\Dibapi.sbr" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Packmsg.cpp
DEP_CPP_PACKM=
"..\Stdafx.h"
"..\Packmsg.h"
"$(INTDIR)\Packmsg.obj" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"
"$(INTDIR)\Packmsg.sbr" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Params.cpp
DEP_CPP_PARAM=
"$(INTDIR)\Dibapi.obj" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"
"$(INTDIR)\Dibapi.sbr" : $(SOURCE) $(DEP_CPP_DIBAP) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Image.cpp
DEP_CPP_IMAGE=
"..\Image.h"
"..\Dibapi.h"
"..\Stdafx.h"
"$(INTDIR)\Image.obj" : $(SOURCE) $(DEP_CPP_IMAGE) "$(INTDIR)"
"$(INTDIR)\Image.sbr" : $(SOURCE) $(DEP_CPP_IMAGE) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Mainfrm.cpp
DEP_CPP_MAINF=
"..\Stdafx.h"
"..\Signer.h"
"..\Mainfrm.h"
"..\Params.h"
"$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"
"$(INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"
!ELSEBIP "$(CFG)" == "Signer - Win32 Debug"
DEP_CPP_MAINF=
"..\Stdafx.h"
"..\Signer.h"
"..\Mainfrm.h"
"$(INTDIR)\Mainfrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"
"$(INTDIR)\Mainfrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"
!ENDIF
# End Source File
#####
SOURCE=.\Myfile.cpp
DEP_CPP_MYFIL=
"..\Stdafx.h"
"..\Dibapi.h"
"$(INTDIR)\Myfile.obj" : $(SOURCE) $(DEP_CPP_MYFIL) "$(INTDIR)"
"$(INTDIR)\Myfile.sbr" : $(SOURCE) $(DEP_CPP_MYFIL) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Mychildw.cpp
DEP_CPP_MYCHIL=
"..\Stdafx.h"
"..\Packmsg.h"
"$(INTDIR)\Mychildw.obj" : $(SOURCE) $(DEP_CPP_MYCHIL) "$(INTDIR)"
"$(INTDIR)\Mychildw.sbr" : $(SOURCE) $(DEP_CPP_MYCHIL) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Readimg.cpp
DEP_CPP_READIM=
"..\Image.h"
"..\Dibapi.h"
"..\Stdafx.h"
"$(INTDIR)\Readimg.obj" : $(SOURCE) $(DEP_CPP_READIM) "$(INTDIR)"
"$(INTDIR)\Readimg.sbr" : $(SOURCE) $(DEP_CPP_READIM) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Signview.cpp
DEP_CPP_SIGNVI=
"..\Stdafx.h"
"..\Packmsg.h"
"$(INTDIR)\Signview.obj" : $(SOURCE) $(DEP_CPP_SIGNVI) "$(INTDIR)"
"$(INTDIR)\Signview.sbr" : $(SOURCE) $(DEP_CPP_SIGNVI) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Signer.cpp
DEP_CPP_SIGNER=
"$(INTDIR)\Packmsg.obj" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"
"$(INTDIR)\Packmsg.sbr" : $(SOURCE) $(DEP_CPP_PACKM) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Signdoc.cpp
DEP_CPP_SIGNDOC=
"..\Stdafx.h"
"..\Dibapi.h"
"$(INTDIR)\Signdoc.obj" : $(SOURCE) $(DEP_CPP_SIGNDOC) "$(INTDIR)"
"$(INTDIR)\Signdoc.sbr" : $(SOURCE) $(DEP_CPP_SIGNDOC) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Align.cpp
DEP_CPP_ALIGN=
"$(INTDIR)\Params.obj" : $(SOURCE) $(DEP_CPP_PARAM) "$(INTDIR)"
"$(INTDIR)\Params.sbr" : $(SOURCE) $(DEP_CPP_PARAM) "$(INTDIR)"
# End Source File
#####
SOURCE=.\Signer.res
DEP_CPP_SIGNER=
"$(INTDIR)\SignerWin32.exe" : $(SOURCE) $(DEP_FILE) $(LINK32_OBJS)
$(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ENDIF

```

[illegible]

```

"$(INTDIR)\SignDoc.obj" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"
"$(INTDIR)\SignDoc.sbr" : $(SOURCE) $(DEP_CPP_SIGND) "$(INTDIR)"
!ENDIF

# End Source File
#####
# Begin Source File
SOURCE-.\Signview.cpp
DEP_CPP_SIGNV-.\
  .\Stdafx.h\
  .\Signer.h\
  .\SignDoc.h\
  .\Signview.h\
  .\Dibapi.h\
  .\Mainfrm.h\
  .\Align.h\
  .\Params.h\
  .\packmsg.h\
  .\Image.h\

"$(INTDIR)\Signview.obj" : $(SOURCE) $(DEP_CPP_SIGNV) "$(INTDIR)"
"$(INTDIR)\Signview.sbr" : $(SOURCE) $(DEP_CPP_SIGNV) "$(INTDIR)"
# End Source File
#####
# Begin Source File
SOURCE-.\Mychildw.cpp
!IF "$(CFG)" == "Signer - Win32 Release"
DEP_CPP_MYCHI-.\
  .\Stdafx.h\
  .\Signer.h\
  .\Mychildw.h\
  .\Params.h\

"$(INTDIR)\Mychildw.obj" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"
"$(INTDIR)\Mychildw.sbr" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"
!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
DEP_CPP_MYCHI-.\
  .\Stdafx.h\
  .\Signer.h\
  .\Mychildw.h\

"$(INTDIR)\Mychildw.obj" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"
"$(INTDIR)\Mychildw.sbr" : $(SOURCE) $(DEP_CPP_MYCHI) "$(INTDIR)"
!ENDIF

# End Source File
#####
# Begin Source File
SOURCE-.\ReadDlg.cpp
!IF "$(CFG)" == "Signer - Win32 Release"
DEP_CPP_READD-.\
  .\Stdafx.h\
  .\Signer.h\
  .\ReadDlg.h\
  .\Params.h\

"$(INTDIR)\ReadDlg.obj" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
"$(INTDIR)\ReadDlg.sbr" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
DEP_CPP_READD-.\
  .\Stdafx.h\
  .\Signer.h\

"$(INTDIR)\ReadDlg.obj" : $(SOURCE) $(DEP_CPP_READD)

```

```

".\readdlg.h"

"$(INTDIR)\ReadDlg.obj" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
"$(INTDIR)\ReadDlg.sbr" : $(SOURCE) $(DEP_CPP_READD) "$(INTDIR)"
!ENDIF

# End Source File
#####
# Begin Source File
SOURCE-.\Signer.def
!IF "$(CFG)" == "Signer - Win32 Release"
!ELSEIF "$(CFG)" == "Signer - Win32 Debug"
!ENDIF

# End Source File
#####
# Begin Source File
SOURCE-.\Align.cpp
"$(INTDIR)\Align.obj" : $(SOURCE) "$(INTDIR)"
"$(INTDIR)\Align.sbr" : $(SOURCE) "$(INTDIR)"
# End Source File
#####
# Begin Source File
SOURCE-.\Pft.cpp
"$(INTDIR)\Pft.obj" : $(SOURCE) "$(INTDIR)"
"$(INTDIR)\Pft.sbr" : $(SOURCE) "$(INTDIR)"
# End Source File
# End Target
# End Project
#####
SIGNVIEW.CPP
// Signview.cpp
// Implementation of the CDibView class
//
//
#include "stdafx.h"
#include "signer.h"

#include "signDoc.h"
#include "signview.h"
#include "dibapi.h"
#include "mainfrm.h"
#include "Align.h"

#include <stratrea.h>
#include <omanip.h>

#ifdef _DEBUG
#define THIS_FILE static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

// CDibview
//
// IMPLEMENT_DYNCREATR(CDibview, CScrollView)
BEGIN_MESSAGE_MAP(CDibview, CScrollView)
    //[[AFX_MSG_MAP(CDibview)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    ON_MESSAGE(WM_DOREALIZE, OnDorealize)
    //]]

```



```
// .....  
// SetViewType()  
// .....  
  
void CDibView::SetViewType(int type)  
{  
    CDibDoc* pDoc = GetDocument();  
    switch (type)  
    {  
        case SIGNED_VIEW:  
            m_viewType = SIGNED_VIEW;  
            // Set the window title.  
            GetParent() ->SetText(GetDocument()->GetTitle() + " -Signed  
                break;  
        case REF_VIEW:  
            m_viewType = REF_VIEW;  
            // Set the window title.  
            GetParent() ->SetText(GetDocument()->GetTitle() + " -Referen  
                break;  
        case ALIGNED_VIEW:  
            m_viewType = ALIGNED_VIEW;  
            // Set the window title.  
            GetParent() ->SetText(GetDocument()->GetTitle() + " -Alignmen  
                break;  
        case STATUS_VIEW:  
            m_viewType = STATUS_VIEW;  
            // Set the window title.  
            GetParent() ->SetText(GetDocument()->GetTitle() + " -Status  
                break;  
        default:  
            // This is an error.  
            afxmessage  
                break;  
    }  
}
```

```

case SIGNED_VIEW:
    m_viewType = SIGNED_VIEW;
    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Signed");
    break;

case REP_VIEW:
    m_viewType = REP_VIEW;
    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Reference");
    break;

case ALIGNED_VIEW:
    m_viewType = ALIGNED_VIEW;
    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Aligned");
    break;

case STATUS_VIEW:
    m_viewType = STATUS_VIEW;
    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status");
    break;

default:
    // This is an error.
    // afxmessage
    break;
}

```

```

case ALIGNED_VIEW:
    m_viewType = ALIGNED_VIEW;
    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Aligned" );
    break;

case STATUS_VIEW:
    m_viewType = STATUS_VIEW;

    // Set the window title.
    GetParent()->SetWindowText(GetDocument()->GetTitle() + " -Status" );
    break;

default:
    // This is an error.
    // afxmessage
    break;
}
)

```

```

case STATUS_VIEW:
    m_viewType = STATUS_VIEW;

    // Set the window title.
    GetParent() ->SetWindowText(GetDocument() ->GetTitle() + " -Status");
    break;

default:
    // This is an error.
    // afxmessage
    break;
}

```

```

        // Set the window title.
        GetParent() -> SetWindowText(GetDocument() -> GetTitle() + " - Status");
        break;

    default:
        // This is an error.
        // atxmessage
        break;
}

```

```

default:
    // This is an error.
    // atxmessage
    break;
}
}

```

—

```

//
// DisplayStatus()
//
void CGridView::DisplayStatus(CDC *pDC)

```

```

CoiDoc* pDoc = GetDocument();
TEXTMETRIC tm;
CString text;
CRect rect;
CFont f;

pDC->GetTextMetrics(&tm);

int col = 20*tm.tmAveCharWidth;
int line = tm.tmEmHeight;
oStream strm;

createStatusStream(strm);

int height;
rect.top = 10;
rect.left = 10;
rect.right = 50 * tm.tmAveCharWidth;
height = pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS | DT_CALCRECT);
rect.bottom = height + 10;
pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS);

// Resize the scrollbars to fit the information it contains.
CSize size = CSize(rect.right+10, rect.bottom);
SetScrollSizes(MM_TEXT, size);

if (m_bDoResizeStatusView)
{
    m_bDoResizeStatusView = FALSE;
    ResizeStatusView(size);
}

// Once we call str(), we must delete the allocated space.
delete strm.str();

return;

```

```
height = pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS | DT_CALCRECT);
rect.bottom = height + 10;
pDC->DrawText(strm.str(), -1, &rect, DT_EXPANDTABS);

// Resize the scrollbars to fit the information it contains.

CSize size = CSize(rect.right+10, rect.bottom);
SetScrollSizes(MM_TEXT, size);

if (m_bDoResizeStatusView)
{
    m_bDoResizeStatusView = FALSE;
    ResizeStatusView(size);
}

// Once we call strm(), we must delete the allocated space.
delete strm.str();

return;
```

```

    {
        m_bDoResizeStatusView = FALSE;
        ResizeStatusView(size);
    }

    // Once we call .str(), we must delete the allocated space.
    delete strm.str();

    return;
}

```

```

// Print crude metric.
strm.precision(4);
strm << "\tBit Estimator Std. Dev.: \t" << pDoc->GetMetric() << "\n\n";
// Print range.
strm << "\tBit Estimator Range: \t" << pDoc->GetRange() << "\n\n";
strm << "\tEmbedded Checksum Read: \t" << (unsigned) pMsg->GetReaderChecksum()
<< "\n\n";
strm << "\tChecksum Calculated: \t" << (unsigned) pMsg->GetComputedReaderChecksum()
<< "\n\n";
}
break;
case SUSPECT_ALIGNED:
AlignStatus a_status = pDoc->GetAlignStatus(); // Get the align status
strm << "Aligned Image Status\n\n";
// Adjust the floating point precision of the stream.
strm.setf(ios::fixed, ios::floatfield);
strm.precision(2);
strm << "\tRotation applied to suspect: \t" << a_status.rotation << "\n\n";
strm << "\tTranslation (X, Y): \t" << a_status.x_trans
<< "\t" << a_status.y_trans << "\n\n";
strm << "\tScaling (X, Y): \t" << a_status.x_scale
<< "\t" << a_status.y_scale << "\n\n";
strm << "\tRefinement: \t" << a_status.refinement << "\n\n";
break;
case SUSPECT_READ:
strm << "Reader Status\n\n";
strm << "\tAssumed Message Length: \t" << pMsg->GetMsgLength() << "\n\n";
strm << "\tRecognized Text: \t" << pMsg->GetRecoveredAsciiMsg() << "\n\n";
strm << "\tAssumed Key: \t" << pDoc->GetSignerParams()->GetKey() << "\n\n";
strm << "\tBump Size: \t" << pDoc->GetSignerParams()->GetBumpSize() << "\n\n";
strm << "\tDetail Gain: \t" << pDoc->GetSignerParams()->GetLutScale() << "\n\n";
// Remove references to "super reader" for now
//if (pDoc->GetSignerParams()->GetSuperReaderPlug())
//strm << "\tAlternative Reader: \t" << "on" << "\n\n";
//else
//strm << "\tAlternative Reader: \t" << "off" << "\n\n";
// Adjust the floating point precision of the stream.
strm.setf(ios::fixed, ios::floatfield);
strm.precision(2);
// Print crude metric.
strm.precision(4);
strm << "\tBit Estimator Std. Dev.: \t" << pDoc->GetMetric() << "\n\n";
// Print range.
strm << "\tBit Estimator Range: \t" << pDoc->GetRange() << "\n\n";
strm << "\tEmbedded Checksum Read: \t" << (unsigned) pMsg->GetReaderChecksum()
<< "\n\n";
strm << "\tChecksum Calculated: \t" << (unsigned) pMsg->GetComputedReaderChecksum()
<< "\n\n";
break;
default:
break;
}
// Add a null terminator (DrawText needs it).
strm << '\0';
}
// ResizeStatusView()
// Resizes the status view frame window. The goal is to not
// move the upper left corner, and to not exceed the bounds of
// the MDI main frame window on the right or left borders.
void CDibView::ResizeStatusView(CSize status_size)
{
const int bar_height = 27; // An empirically derived kludge

```

SIGNVIEW.H

```
// signview.h : interface of the CDbView class
//
#include <strstream>

// Here I define the different types of views.
#define UNKNOWN_VIEW -1
#define SIGNED_VIEW 1
#define ORIGINAL_VIEW 2
#define SNOWY_VIEW 3
#define STATUS_VIEW 4
#define REF_VIEW 5
#define ALIGNED_VIEW 6

// reference image for alignment
// image after alignment completed

class CDbView : public CScrollView
{
public:
    CDbView();
    DECLARE_DYNCREATE(CDbView)

// Attributes
public:
    CDbDoc* GetDocument()
    {
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDbDoc)));
        return (CDbDoc*) m_pDocument;
    }

private:
    int m_viewType;
    BOOL m_bThisViewActive;
    BOOL m_bDoResizeStatusView;

// Operations
public:

// Implementation
public:
    virtual ~CDbView();
    virtual void OnDraw(CDC* pDC); // overridden to draw this view

    virtual void OnInitialUpdate();
    virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
        CView* pDeactivateView);
    void SetViewType(int type);
    int GetViewType(void) {return m_viewType;}
    BOOL IsViewActive(void) {return m_bThisViewActive;}

    void DoResize(void) {m_bDoResizeStatusView = TRUE;}
    void ResizeStatusView(CSize status_size);

// I need OnFilePrint to be accessible from outside.
    void OnFilePrint(void) {CScrollView::OnFilePrint();}

    void CreateStatusStream(ostream& strm);

// Printing support
protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);

private:
    HDIB GetHDIB(void);
    void CDbView::DisplayStatus(CDC *pDC);

// Generated message map functions
protected:
    ///({AFX_MSG(CDbView)
    afx_msg void OnEditCopy();
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
    afx_msg void OnEditPaste();
    afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
    afx_msg LRESULT OnDoRealize(HWND wParam, LPARAM lParam); // user message
    afx_msg void OnViewSigned();
    afx_msg void OnViewUnsigned();
    afx_msg void OnViewStatus();
    afx_msg void OnUpdateViewSigned(CCmdUI* pCmdUI);
    afx_msg void OnUpdateViewSnowyImage(CCmdUI* pCmdUI);
    afx_msg void OnUpdateViewStatus(CCmdUI* pCmdUI);
    afx_msg void OnUpdateViewUnsigned(CCmdUI* pCmdUI);
    ///})AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
```

SNOWTMP.CPP

```
////////////////////////////////////
// My experimental member function which
// builds a snowy image in place.
//
////////////////////////////////////

void CDbDoc::MakeSnow(void)
{
    int cxDIB, cyDIB;
    long num_pixels, num_colors;
    LPSTR lpDIB, lpSnowyDIB; // pointer to BITMAPINFOHEADER
    LPBITMAPINFOHEADER lpDIBHdr, lpSnowyDIBHdr;
    LPSTR lpDIBBits; // pointer to DIB bits
    char __huge *src_data, *dest_data; // huge ptrs for copying the image.

    HDIB hUnsignedDIB = GetHDIB();
    if (hUnsignedDIB == NULL)
        return;

// Create space for the unsigned DIB for the snowy image.
    m_hSnowyDIB = (HDIB)::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, m_dwTotalDIBSize);
    if (m_hSnowyDIB == 0)
        return;

// Here I follow the similar code in PaintDIB() of dibapi.cpp
    lpDIB = (LPSTR)::GlobalLock((HGLOBAL) hUnsignedDIB);
    lpSnowyDIB = (LPSTR)::GlobalLock((HGLOBAL) m_hSnowyDIB);
    src_data = (char__huge *) lpDIB;
    dest_data = (char__huge *) lpSnowyDIB;

// Copy the BITMAPINFOHEADER, palette, and actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }

    lpDIBHdr = (LPBITMAPINFOHEADER) lpDIB; // Ptr to bitmap info hdr at start of dib.

// Get ptr to the snowy dib header space, and copy header into it.
    lpSnowyDIBHdr = (LPBITMAPINFOHEADER) lpSnowyDIB;
    *lpSnowyDIBHdr = *lpDIBHdr;

    lpDIBBits = ::FindDIBBits(lpDIB);
    lpSnowyDIBBits = ::FindDIBBits(lpSnowyDIB);
    src_data = (char__huge *) lpDIBBits;
    dest_data = (char__huge *) lpSnowyDIBBits;

// Copy the actual image byte data.
    for (image_byte = 0; image_byte < m_dwTotalDIBSize; image_byte++)
    {
        dest_data++ = src_data++;
    }

    cxDIB = (int)::DIBWidth(lpDIB); // X size of DIB
    cyDIB = (int)::DIBHeight(lpDIB); // Y size of DIB
    num_pixels = (long) cxDIB * cyDIB;
    num_colors = ::DIBNumColors(lpDIB);
    if (lpDIBHdr->biCompression != 0)
    {
        TRACE("Can't cope with compressed image (compression = %d)\n",
            ::GlobalUnlock((HGLOBAL) hUnsignedDIB);
        return;
    }
    TRACE("width = %d, height = %d, num_pixels = %d\n", cxDIB, cyDIB, num_pixels);
    TRACE("num_colors = %d\n", num_colors);
    if (num_colors == 0 || num_colors == 16)
    {
        //////////////////////////////////////
```



```

TRACE("At this time, only build snow image for 8 bit images\n");
::GlobalUnlock((HGLOBAL) hUnsignedDIB);
return;
}

if (num_colors == 256)
{
    CoKey coKey(1, (BITMAPINFO *) lpDIBHdr, lpDIBBits);
}

::GlobalUnlock((HGLOBAL) hUnsignedDIB);

}

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.

// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"

// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992 Microsoft Corporation
// All rights reserved.

// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and Microsoft
// QuickHelp and/or WinHelp documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
#include <afxwin.h> // MFC core and standard components

//////////////////////////////////////
// FILE: Sign_public.cpp
//////////////////////////////////////
// DESCRIPTION:
// Core signing functions of the public digimarc technology.
// Started late April 1996
// Copyright (C) 1996 Digimarc Corporation, all rights reserved.
//////////////////////////////////////
#include "sign.h"
#include <math.h>
#include "stdafx.h"

#define SIGNATURE_BLOCK_DIMENSION 128
#define HIGHEST_GRAY_VALUE 255
#define GRID_MINIMUM_GAIN -0.5
#define RAD_DOG 0.33
#define GREEN_DOG 0.34
#define BLUE_DOG 0.33

// this function simply loads the floating point values of the bumps for a given "bump raster line"
// the output of this function (the bump array) should be roughly similar no matter
// what the bump size is or whether you're dealing with color or B&W
// REMEMBER: this function pads the ends on each side with one extra bump
int load_bump_array(
    float *bump, // floating point bump array to be filled (output)
    unsigned char *data, // input pixel data
    long xdim, // number of bumps in this row (not pixels), add 2 for output
    long zdim, // number of channels
    long bump_size, // pixels per bump
    long jump_x, // number of raw pixels between (xdim*bump_size) and entire image array x
    dimension
    into the first and last place
    ) {
    unsigned char *pdata;
    long i,j,k;
    float *bump, bump_squared = (float)bump_size * (float)bump_size;

    pdata = data;
    if (overflow) bump = bump+1;
    else bump = bump;
    if (zdim == 1) { // single channel
        if (bump_size == 1) {
            for (j=0; j<xdim; j++) *pbump++ = (float) * (pdata++);
        }
        else if (bump_size == 2) {
            // zero out bump array
            memset(bump, 0, (xdim+2)*sizeof(float));
            for (i=0; i<2; i++) {
                if (overflow) pbump = bump+1;
                else pbump = bump;
                for (j=0; j<xdim; j++) {
                    *pbump++ = (float) * (pdata++);
                    *pbump++ += (float) * (pdata++);
                }
                pdata += jump_x;
            }
            if (overflow) pbump = bump+1;
            else pbump = bump;
            for (i=0; i<xdim; i++) *pbump++ = (float) * (pdata++);
        }
        else { // zero out bump array
            memset(bump, 0, (xdim+2)*sizeof(float));
            for (i=0; i<bump_size; i++) {
                if (overflow) pbump = bump+1;
                else pbump = bump;
                for (j=0; j<xdim; j++) {
                    for (k=0; k<bump_size; k++) *pbump++ = (float) * (pdata++);
                    pbump++;
                }
                pdata += jump_x;
            }
            if (overflow) pbump = bump+1;
            else pbump = bump;
            for (i=0; i<xdim; i++) *pbump++ = (float) * (pdata++);
        }
    }
    else { // multi-channel, assume ONLY RGB and three channels at present
        float red = (float) RED_DOG, green = (float) GREEN_DOG, blue = (float) BLUE_DOG;
        if (bump_size == 1) { // this case is split off only for a % speed increase in
            execution
            for (j=0; j<xdim; j++) {
                *pbump = red * (float) * (pdata++); // gimme an R
                *pbump += green * (float) * (pdata++); // gimme a G
                *pbump++ += blue * (float) * (pdata++); // gimme a B
            }
        }
        else {
            // zero out bump array
            memset(bump, 0, (xdim+2)*xdim*sizeof(float));
            for (i=0; i<bump_size; i++) {
                if (overflow) pbump = bump+1;
                else pbump = bump;
                for (j=0; j<xdim; j++) {
                    for (k=0; k<bump_size; k++) {
                        *pbump++ = red * (float) * (pdata++); // gimme an R
                        *pbump++ += green * (float) * (pdata++); // gimme a G
                        *pbump++ += blue * (float) * (pdata++); // gimme a B
                    }
                    pbump++;
                }
                pdata += zdim * jump_x;
            }
            if (overflow) pbump = bump+1;
            else pbump = bump;
            for (i=0; i<xdim; i++) *pbump++ = (float) * (pdata++);
        }
    }
    // fill the end two values
    if (overflow) {
        bump[0] = bump[1];
        bump[xdim+1] = bump[xdim];
    }
}

```

```

    return(1);
}

// load_funky_lut()
// This function loads the scaling factor based on minimum linear funkiness
// int load_funky_lut(float *funky_lut) // explicitly written for 8 bit
{
    int i, status=1, detail_start, detail_stop;
    float length;
    float scale = (float)1.0;
    detail_start = 1;
    detail_stop = 50;
    length = (float)detail_stop - (float)detail_start;
    for(i=0; i<detail_start; i++) funky_lut[i] = (float)0.0;
    {
        for(i=detail_start; i<detail_stop; i++)
        {
            funky_lut[i] = scale*((float)(i-detail_start)/length);
        }
        for(i=detail_stop; i<512; i++) funky_lut[i] = funky_lut[detail_stop-1];
    }
    return(status);
}

// This function associates a given row and column value of a bump in the
// standard signature block with A) the bit plane of the message associated with the bump,
// output in the 'message_bit_lut' variable array, and B) whether the '1' direction is up,
// XOR lut=1, or down, XOR lut=0
// IMPORTANT: this also takes care of the basic XOR'ing operation between the message and
// the underlying code pattern (invert, don't invert)
// int load_standard_message_block_lut(
//     unsigned char *message, // if this is NULL, return the un XOR'ed array (for reading)
//     long message_length,
//     unsigned char *control_message, // this is the separate "always gotta be there" message
//     long control_message_length, // its length
//     short *message_bit_lut,
//     unsigned char *XOR_lut,
//     long read_or_write
// ) {
    // This is a crude first version... April 1996
    // we're going with 16 control bits, and in this demo, we'll use all of them
    // to describe the raw message length as a short unsigned int
    // int *length_table = new int(15);
    // int *blocks = new int(15);
    // int *yblocks = new int(15);
    int length_table[] = { 15, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512, 768, 1024, 1536, 3072 };
    int xblocks[] = { 8, 8, 4, 8, 4, 2, 4, 2, 4, 2, 4, 2, 1, 1, 1 };
    int yblocks[] = { 8, 8, 8, 8, 4, 8, 4, 2, 4, 2, 4, 2, 1, 2, 1 };
    // find which length in the length table is next highest over current message_length
    long index=0;
    while( length_table[index] < message_length ) {
        index++;
    }
    long length = (SIGNATURE_BLOCK_DIMENSION/21)/xblocks[index]; // length in bumps
    long ylength = (SIGNATURE_BLOCK_DIMENSION/21)/yblocks[index];
    long current_bit = kfoo, lfoo;
    long jump = SIGNATURE_BLOCK_DIMENSION;
    short actual_bit;
    long one;
    long i, j, k, l;
    short *message_bit;
    unsigned char *pxor;
    for(i=0; i<yblocks[index]; i++) {
        current_bit = 11 + j*length; // this is
        // simply a "mixing agent" so that given bit planes
        // don't segregate around edges (come up with a better way please please
        // the following uses the
        // 1 0
        // 0 1
        // formula of local bumps associated with a given bit plane, hence the 2's
        // floating around
        for(k=0; k<ylength; k++) {
            // reset the pointers
            message_bit = &message_bit_lut[2*j*length + 2*(i*ylength+k)*jump];
            ptweak++;
        }
    }
}

pxor = sxor_lut[2*j*length + 2*(i*ylength+k)*jump];
kfoo = (k+6)%8;
for(l=0; l<length; l++) {
    lfoo = (l+6)%8;
    if(kfoo<4 && lfoo<4) { // this is the 16 bit control code region
        } actual_bit = (short)(message_length + kfoo*4 + lfoo);
    } else { // this is the embedded data region
        actual_bit = (short)(current_bit + message_length);
        current_bit++;
    }
    *message_bit = *(message_bit+1) = *(message_bit+jump) =
        *(message_bit+jump+1) = actual_bit;
    pmessage_bit+=2;
    if(read_or_write) one = 1;
    else {
        if(actual_bit >= message_length) {
            if(control_message[actual_bit-message_length]) one = 1;
            else one = 0;
        }
        else {
            if(message[actual_bit]) one = 1;
            else one = 0;
        }
    }
    if(one) {
        *pxor = 1;
        *pxor+1 = 0;
        *pxor+jump = 0;
        *pxor+jump+1 = 1;
    }
    else {
        *pxor = 0;
        *pxor+1 = 1;
        *pxor+jump = 1;
        *pxor+jump+1 = 0;
    }
    pxor+=2;
}
}
}
//delete () length_table;
//delete () xblocks;
//delete () yblocks;
return(1);
}

int load_output_array(
    float *tweak,
    unsigned char *data_out,
    unsigned char *data,
    long xdim,
    long ydim,
    long jump_x,
    long jump_y
) {
    unsigned char *pdata, *pdata_out;
    int i, j, k, temp;
    float *ptweak, half = (float)0.5;
    pdata = data;
    ptweak = tweak;
    pdata_out = data_out;
    if(xdim == 1) { // single channel
        if(bump_size == 1) {
            for(j=0; j<xdim; j++) {
                temp = (int)((float)*(pdata++) + *(ptweak++) + half);
                if(temp<0) *pdata_out++=0;
                else if(temp>HIGHEST_GREY_VALUE) *pdata_out++=HIGHEST_GREY_VALUE;
                else *pdata_out++ = (unsigned char)temp;
            }
        }
        else {
            for(i=0; i<bump_size; i++) {
                ptweak = tweak;
                for(j=0; j<xdim; j++) {
                    temp = (int)((float)*(pdata++) + *(ptweak++) + half);
                    if(temp<0) *pdata_out++=0;
                    else if(temp>HIGHEST_GREY_VALUE) *pdata_out++=HIGHEST_GREY_VALUE;
                    else *pdata_out++ = (unsigned char)temp;
                }
                ptweak++;
            }
        }
    }
}

```

```

pdata += jump_x;
pdata_out += jump_x;
}
}
else { // multi-channel, assume ONLY RGB and three channels at present
float red = (float)RED_DOG, green = (float)GREEN_DOG, blue = (float)BLUE_DOG;
float red_ratio, green_ratio, blue_ratio, lum_ratio, lum_zero = (float)0.1;
if (bump_size == 1) {
for (j=0; j<xdim; j++) {
lum = red * (float)*pdata + green * (float)*(pdata+1) + blue * (float)*(pdata+2);
if (lum > zero) {
red_ratio = (float)*(pdata+1) / lum;
green_ratio = (float)*(pdata+2) / lum;
blue_ratio = (float)*(pdata+3) / lum;
}
else {
red_ratio = green_ratio = blue_ratio = (float)1.0;
pdata+=3;
}
lum += *ptweak++;
// red
temp = (int)( lum * red_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = (unsigned char)HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
// green
temp = (int)( lum * green_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = (unsigned char)HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
// blue
temp = (int)( lum * blue_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = (unsigned char)HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
}
}
else {
for (i=0; i<bump_size; i++) {
ptweak = tweak;
for (j=0; j<xdim; j++) {
for (k=0; k<bump_size; k++) {
lum = red * (float)*pdata + green * (float)*(pdata+1) + blue *
(float)*(pdata+2);
if (lum > zero) {
red_ratio = (float)*(pdata+1) / lum;
green_ratio = (float)*(pdata+2) / lum;
blue_ratio = (float)*(pdata+3) / lum;
}
else {
red_ratio = green_ratio = blue_ratio = (float)1.0;
pdata+=3;
}
lum += *ptweak;
// red
temp = (int)( lum * red_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
// green
temp = (int)( lum * green_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
// blue
temp = (int)( lum * blue_ratio + half );
if (temp < 0) *pdata_out++ = 0;
else if (temp > HIGHEST_GREY_VALUE) *pdata_out++ = HIGHEST_GREY_VALUE;
else *pdata_out++ = (unsigned char)temp;
}
}
ptweak++;
}
pdata += jump_x * zdim;
pdata_out += jump_x * zdim;
}
}
return(1);
}

// core_sign_public_generation_11
//
// problem has been reduced to basic block unit;
// the only special case is when xdim and/or ydim are not extended to full block size
//
int core_sign_public_generation11 {

```

```

unsigned char *data, // pointer to upper left corner of image block
long xdim, // absolute pixel dimension of current block
long Original_xdim, // absolute pixel dimension of entire original image or passed array
long ydim, // absolute pixel dimension of current block
long zdim, // number of channels, e.g. 3 for RGB
long bump_size, // message length
long message_length, // message length
short message_bit_lut, // this can be economized and reduced by 8 by using bitwise
// packing if done in another here
float *detail_lut,
float *subliminal_grid,
float *subliminal_out, // NULL if data is to be put back into input array
float global_gain,
float asymetric_gain,
float *funky_lut
} {
long jump_x = Original_xdim - xdim; // this is the pointer offset for jumping rows
unsigned char *pdata_out;
long i, j;
float *x2, *x3, *x4, *pbump, local_average_gain, detail_gain, diff;
float *subliminal_grid, lum_gain, asym_gain, funky_gain;
short *bit;
unsigned char *pxor;
double dtweak, bottomtweak;

// set pdata_out based on (in place) versus new output array
if (data_out == NULL) pdata_out = data;
else pdata_out = data_out;

// calculate bitwise bias between original image, (optionally degraded by common-model
// distortion), and each bit of the message; this will be used for differential gain of
// the bit planes to help "struggling" bits
float *bit_bias = new float(message_length);
for (i=0; i<message_length; i++) bit_bias[i] = (float)1.0;
// read_block_signature {
// convert_read_to_bias {
// dive into main loop
//
Main loop version 1 works in the following way. It is designed so that it can
create a logged version of the output in order to support either case of: A) where
the input data array is replaced with the output array (in place), or B) where the
"data_out" pointer is not null and is the actual output array.
--- THIS PARTICULAR VERSION EXPECTS CASE B ---

The main loop essentially operates bump by bump. It determines the local overall
gain that should be applied to the given bump, then tweaks the individual pixel(s)
of the output bump and stores in the temporary array which is later written out into
the ultimate output array.

long xbumpdim = xdim/bump_size; // calling routine guaranteed this would never have a
remainder
long ybumpdim = ydim/bump_size;
// create initial bump arrays
int xbumpsize = xbumpdim*2; // adding '2' allows us to not worry about edges in core loops
float *bump0 = new float[xbumpsize];
float *bump1 = new float[xbumpsize];
float *bump2 = new float[xbumpsize];
// load row 1 and row 2 (with row 0 data) for the first process step
// load_bump_array should copy elements 0 and 1 with data bump 0
// and elements xbumpdim and xbumpdim+1 with data bump xbumpdim-1
load_bump_array(bump1, data, xbumpdim, zdim, bump_size, jump_x, 1);
load_bump_array(bump2, bump1, xbumpsize*sizeof(float));
// create tweak array for each raster of bumps
float *tweak = new float[xbumpdim];
float *ptweak;
float f1 = (float)1.0;
float f4 = (float)4.0;

for (i=0; i<ybumpdim; i++) {
// in order to avoid modulo housekeeping later on, copy the arrays downward
// (as they are small too)
memcpy(bump0, bump1, xbumpsize*sizeof(float));
memcpy(bump1, bump2, xbumpsize*sizeof(float));
if (i != (ybumpdim-1)) { // load next bump row array
load_bump_array(bump2, data, (i+1)*bump_size*Original_xdim, xbumpdim, zdim, bump_size, jump_x,
1);
}
else { // leave bump2 alone
p1 = bump0+1;
p2 = bump1;
p3 = bump2+1;
p4 = bump1+2;
pbump = bump1+1;
psubliminal_grid = subliminal_grid[i*SIGNATURE_BLOCK_DIMENSION];
ptweak = tweak;
}
}
}

```

```

time
pbic = !message_bit_lut[!(SIGNATURE_BLOCK_DIMENSION)];
pxor = !xor_lut[!(SIGNATURE_BLOCK_DIMENSION)];
for(j=0;j<xbumpdim;j++){ // this is the heart of the signing code and process, one bump at a

```

/* Here's the deal: (Written 4/26/96)

The goal of the signing process, beyond simply functioning, is to maximize the "numeric detectability" of an embedded signature while meeting some form of fixed "visibility/acceptability threshold" set by a given user/creator.

In service to design toward this goal, imagine the following three axis parameter space, where two of the axes are only half-axes (positive only), and the third is a full axis (both negative and positive). This set of axes define two of the usual eight octal spaces of euclidean 3-space. As things refine and "deservedly separable" parameters show up on the scene (such as "extended local visibility metrics"), then they can define their own (generally) half-axis and extend the following example beyond three dimensions.

The signing design goal becomes optimally assigning a "gain" to a local bump based on its coordinates in the above defined space, whilst keeping in mind the basic needs of doing the operations fast in real applications. To begin with, the three axes are the following. We'll call the two half axes x and y, while the full axis will be z.

The x axis represents the luminance of the singular bump. The basic idea is that you can squeeze a little more energy into bright regions as opposed to dim ones. It is important to note that when true "psycho-linear" device independent luminance values (pixel DN's) come along, this axis might become superfluous, unless of course if the luminance value couples into the other operative axes (e.g. Cxy). For now, this is here as much due to the sub-optimality of current quasi-linear luminance coding.

The y axis is the kitchen sink of "local hiding potential" of the neighborhood within which the bump finds itself. The basic idea is that flat regions have a low hiding potential since the eye can detect subtle changes in such regions, whereas complex textured regions have a high hiding potential. Long lines and long edges tend toward the lower hiding potential since "breaks and chopiness" in nice smooth long lines are also somewhat visible, while shorter lines and edges, and mosaics thereof, tend toward the higher hiding potential. These latter notions of long and short are directly connected to processing time issues, as well to issues of the engineering resources needed to carefully quantify such parameters. Developing the working model of the y-axis will inevitably entail one part theory to one part picky-artist-empiricism. As the parts of the hodge-podge y-axis become better known, they can splinter off into their own independent axes if it's worth it.

The z-axis is the "with or against the grain" axis which is the full axis - as opposed to the other two half-axes. The basic idea is that a given input bump has a pre-existing bias relative to whether one wishes to encode a '1' or a '0' at its location, which to some non-trivial extent is a function of the reading algorithms which will be employed, whose (bias) magnitude is semi-correlated to the "hiding potential" of the y-axis, and, fortunately, . . . can be used advantageously as a variable in determining what magnitude of a tweak value is assigned to the bump in question. The concomitant basic idea is that when a bump is already your friend, or even your friend in a big way, then why mess with it much, whereas when it is your enemy or a big time enemy, then you want to squash it like a four year old discovering how flat slugs can get underfoot. The really cool thing here is that, in general, the latter squashing operation tends more toward a local blurring operation as opposed to a local sharpening operation, and thus has somewhat less visibility per numeric tweak unit.

The above general description of the problem should suffice for many years. Clearly adding in chrominance issues will expand the definitions a bit, leading to a bit more signature bang for the visibility, and human visibility research which is applied to the problem of compression can equally be applied to this area but for diametrically opposed reasons. Pacinating possibilities truly. But alas, I am required to crank out some pot-shot first system which needs must neglect vast areas of the above general arenas. Here are its principles.

For speed's sake, local hiding potential will be calculated only based on a 3 by 3 neighborhood of pixels, the center one being signed and its eight neighbors. Beyond speed issues, there is also no data or coherent theory to support anything larger as well. The design issue boils down to coming to the y-axis visibility thing, how to couple the luminance into this, and a little bit on the friend/enemy asymmetry thing. My guiding principles to start are simply to make a flat region zero a "static pure maxima or minima region a 1.0" or the highest value, and to have "local lines", "smooth slopes", "saddle points" and whatnot fall out somewhere in between. In other words, let's pull out the darts and throw a few and see if any land on the board.

The following code has six basic parameters that will be used:

- 1) luminance
- 2) difference from local average
- 3) the asymmetry factor (with or against the grain)
- 4) minimum linear funkiness factor (our crude attempt at flat v. lines v. maxima)
- 5) bit plane bias factor

- 6) global gain (the user's single top level gain knob)
- Even this list above can get complicated in their inter-relations and especially in our current lack of experimental data to support various specific formulas.

- 1) luminance is straightforward
- 2) difference from local average is also, and is rather important to our first generation stuff since it will directly eb involved in readability signatures (assuming we don't get fancy phase-only reading algorithms going).
- 3) the asymmetry factor is a single scalar applied to the "against the grain" side of the difference axis of number 2 directly above, as well and being modified by the minimum linear funkiness factor below. [Certainly it can eventually become a function of other variables if and when data and theory supports such].
- 4) the minimum linear funkiness factor is admittedly crude but it should be of some service even in a 3 by 3 neighborhood setting. The idea is that true 2D local minima and maxima will be highly perturbed along each of the four lines travelling through the center pixel of the 3 by 3 neighborhood, while a "visual line or edge will tend to flatten out at least one of the four linear profiles. [The four linear profiles are each 3 pixels in length, i.e., the top left pixel - center - bottom right; the top center - center - bottom center; the top right - center - bottom left; the right center - center - left center]. Let's choose some metric of "funkiness" or entropy as applied to three pixels in a row, perform this on all four linear profiles, then choose the minimum value for our ultimate parameter to be used as our 'y-axis'. Cheers to she or he who will take all of this to the next levels of refinement.
- 5) The bit plane bias factor is an interesting creature with two faces, the pre-emptive face and the post-emptive face. In the former, you simply "read" the unsigned image and see where all the biases fall out for all the bit planes, then simply boost the "global gain" of the bit planes which are, in total, going against your desired message, and leave the others alone or even slightly lower their gain. In the post-emptive modification, you churn out the whole signing process replete with the pre-emptive bit plane bias and the other 5 parameters listed here, and then you e.g. run the signed image through heavy JFS compression AND model the "gestalt distortion" of line screen printing and subsequent scanning of the image, and then you read the image and find out which bit planes are struggling or even in error, you appropriately beef up the bit plane bias, and you run through the process again. If you have good data driving the beefing process you should only need to perform this step once, or, you can easily Van-Cittertize the process (arcane reference to regenerate the process with some damping factor applied to the tweaks).
- 6) Finally, there is the global gain. The goal is to make this single variable be the top level "intensity knob" that the slightly curious user can adjust if they want to. The very curious user can navigate down advanced menus to get their epiformal hands on the other five variables here, and who knows what others in the future.

when, that's the most commenting I've ever done, I must be getting old or maybe I'm just realizing it would be nice to leave a signpost or two in this first dart throwing.

```

// get luminance gain
lum_gain = luminance_lut[ (int)*pbump ];

// find current differential between bump value and local average
// this one can generally make use of Inter-DN lut's:
local_average = *p1 + *p2 + *p3 + *p4;
diff = *pbump * f4 - local_average;
detail_gain = detail_lut[ (int)( fabs( (double)diff ) ) ];

// now calculate tweak based first on message, include asymmetric gain
if( *pxor ){
    if(diff<0.0) asym_gain = asymmetric_gain;
    else asym_gain = fl;
    *ptweak = fl; // slip this one in here
}
else {
    if(diff>0.0) asym_gain = asymmetric_gain;
    else asym_gain = fl;
    *ptweak = -fl;
}

// funky time: minimum linear funkiness factor
// line 1
bottomfunk = fabs((double)( *pbump - *(p1-1) )) + fabs((double)( *pbump - *(p3+1) ));
// line 2
dtemp = fabs((double)( *pbump - *p1 )) + fabs((double)( *pbump - *p3 ));
if(dtemp < bottomfunk) bottomfunk = dtemp;
// line 3
dtemp = fabs((double)( *pbump - *(p1+1) )) + fabs((double)( *pbump - *(p3-1) ));
if(dtemp < bottomfunk) bottomfunk = dtemp;
// line 4
dtemp = fabs((double)( *pbump - *p2 )) + fabs((double)( *pbump - *p4 ));
if(dtemp < bottomfunk) bottomfunk = dtemp;
funky_gain = funky_lut[ (int)bottomfunk ];

```

